Workshop on Verification and Theorem Proving for Continuous Systems (NetCA Workshop 2005)

Kanovich, Max; White, Graham; Gottliebsen, Hanne; Oliva, Paulo

ISSN 1470-5559

Workshop on Verification and Theorem Proving for Continuous Systems (NetCA Workshop 2005)

Editors: Max Kanovich, Graham White, Hanne Gottliebsen and Paulo Oliva

Queen Mary
University of London
Department of Computer Science

RR-05-06

August 26 2005
Oxford, UK

Workshop on Verification and Theorem Proving for Continuous Systems (NetCA Workshop 2005)

Editors: Max Kanovich, Graham White, Hanne Gottliebsen and Paulo Oliva

Abstract: The UK Network in Computer Algebra (NETCA) organised a workshop on various aspects of continuous systems verification, mainly with the focus on theorem proving. The workshop provides a forum for work in progress on new and emerging areas to be presented and discussed, and experiences to be shared.

Contents **Page**

An algebraic analysis approach to mathematical system theory

A. Quadrat,

INRIA Sophia Antipolis,
CAFE project,
2004 Route des Lucioles BP 93,
06902 Sophia Antipolis Cedex, France.
Alban.Quadrat@sophia.inria.fr
http://www-sop.inria.fr/cafe/Alban.Quadrat/index.html

This work has been done in collaboration with:

F. Chyzak,
INRIA Rocquencourt,
ALGO project,
Domaine de Voluceau, BP 105,
78153 Le Chesnay Cedex, France,
frederic.chyzak@inria.fr,

D. Robertz,
Lehrstuhl B für Mathematik RWTH - Aachen,
Templergraben 64, Aachen, Germany,
daniel@momo.math.rwth-aachen.de.

Keywords: Mathematical system theory, control theory, algebraic analysis, effective algebra, symbolic computation.

In the seventies, the study of transfer matrices of time-invariant linear systems of ordinary differential equations (ODEs) led to the development of the *polynomial approach* [19, 21, 41]. In particular, the univariate polynomial matrices play a central role in this approach (e.g., Hermite, Smith and Popov forms, invariant factors, primeness, Bézout/Diophantine equations).

In the middle of the seventies, while generalizing linear systems defined by ODEs to differential time-delay systems, ODEs with parameters, 2-D and 3-D filters..., one had to face the case of systems described by means of matrices with entries in multivariate commutative polynomial rings. All these new systems were called 2-D or 3-D linear systems and, more generally,

n-D systems or *multidimensional linear systems* with constant coefficients [4, 15]. It was quickly realized that no canonical forms such as Hermite, Smith and Popov forms existed for polynomial matrices with two and three variables (i.e., with entries in $k[x_1, x_2, x_3]$, where k is a field such as $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$). Moreover, more than only one type of primeness was needed in order to classify n-D systems (e.g., factor/minor/zero primeness [45, 46]). Hence, it is not very much surprising that, in the eighties, *Gröbner bases* were introduced in the study of multidimensional linear systems with constant coefficients [4, 15]. A Gröbner basis defines normal forms for polynomials with respect to a certain monomial order of the variables x_i [2, 16, 22]. Given a Gröbner basis, there is a simple algorithm to effectively compute these normal forms. In many ways, the computation of these normal forms can be seen as an extension of the Gaussian elimination algorithm to commutative polynomial rings [2, 16].

In a pioneering work, R. E. Kalman developed a *module-theoretic* approach to time-invariant ordinary differential linear systems [20]. In his PhD thesis under the supervision of R. E. Kalman, Y. Rouchaleau considered Kalman-type systems where the entries of (A, B, C, D) belong to a commutative ring. In particular, he studied their structural properties using module theory. Such systems are nowadays called *systems over rings* and they have been considerably studied in the literature [40] since. An extension of the *geometric approach* [43] to linear systems over rings has also been recently developed [1, 12, 17, 18]. Using effective algebra methods (Gröbner bases, *characteristic sets*), the computational aspects of the systems over rings (e.g., differential time-delays systems) were firstly studied by L. Habets in [17, 18].

In the nineties, U. Oberst developed a general module-theoretic approach to multidimensional linear systems with constant coefficients [27]. Using B. Malgrange's approach [23], in which a *finitely presented D-module M* is associated with a linear system of equations over a polynomial ring D, he showed how some structural properties of the system corresponded to algebraic properties of the D-module M. He then was able to develop a complete duality between his module-theoretic approach and the *behavioural approach* developed by J. C. Willems [29]. Based on U. Oberst's ideas, the behavioural approach to multidimensional linear systems has been successfully developed in the recent years. See [29, 28, 35, 44, 46] and the references therein.

Within a similar module-theoretic approach, the concepts of *flatness* and *π-freeness* were introduced in [14, 25] for differential time-delay linear systems with constant coefficients. As it is shown in [25, 26] on different concrete examples, the detection of such structural properties is important for the study of the motion planning problem. In the behavioural approach, the concept of flatness corresponds to the existence of an *observable image representation* for the multidimensional system [31, 39].

In the same years as [27], J.-F. Pommaret studied under-determined

systems of partial differential equations (PDEs) coming from mathematical physics and differential geometry (e.g., elasticity, electromagnetism, hydrodynamics, general relativity). See also [3]. In particular, he showed how his mathematical approach was a generalization of U. Oberst's module-theoretic approach for multidimensional (linear) systems with varying coefficients. See [30] for more details and references. In particular, the problem of checking whether or not a multidimensional linear system described by PDEs with varying coefficients could be formally parametrized was solved within the theory of differential operators. Moreover, the work of M. Fliess on linear systems defined by ODEs with variable coefficients also illustrated the need to pass from the commutative polynomial viewpoint to the non-commutative one [13].

Based on B. Malgrange's approach [23], *algebraic analysis* has been developed in mathematics in order to study general linear systems of PDEs with variable coefficients using module theory, algebraic geometry, homological algebra and functional analysis. Algebraic analysis has recently been introduced in control theory in [37] in order to study multidimensional linear systems defined by PDEs with varying coefficients. This approach extends the one previously developed by J.-F. Pommaret for underdetermined linear systems of PDEs. In particular, using the formal theories of PDEs (Spencer's, Riquier-Janet's theories), it was shown in [30, 31, 32, 33, 36, 37] how some structural properties of systems could be checked by means of effective algorithms.

Finally, using the homological algebra approach developed in [37], we have recently shown in [9, 11, 38, 39] how the previous results could be generalized to some classes of multidimensional linear systems with varying coefficients encountered in the literature (e.g., ODEs, PDEs, differential time-delay systems, multidimensional discrete systems, partial differential delay systems). In order to do that, the concept of multidimensional linear systems over *Ore algebras* was introduced in [9, 11]. An Ore algebra is a ring of non-commutative polynomials in functional operators with polynomial or rational coefficients [5, 6, 7, 24]. The characterization of algebraic structural properties such as, for instance, controllability, parametrizability and flatness were obtained.

The recent progress of Gröbner bases over Ore algebras (i.e., over some classes of non-commutative polynomial rings) [5, 6, 7, 22] allows us to effectively test the algebraic properties of general multidimensional linear systems (e.g., controllability, observability, parametrizability, flatness) and compute different types of parametrizations and to propose feedback laws (motion planning, tracking, Bézout equations, optimal control).

In this presentation, we shall develop the following methodology for the study of multidimensional linear systems over Ore algebras (see also [11]):

1. A linear system is defined by means of a $(q \times p)$-matrix R with entries in

an Ore algebra D, i.e., it corresponds to a system of linear equations $R\,z = 0$, where z is composed of the system variables (e.g., inputs, outputs, states, latent variables).

2. We associate the finitely presented left D-module $M = D^{1\times p}/(D^{1\times q}\,R)$ with the system $R\,z = 0$.

3. We develop a dictionary between the structural properties of the system and the properties of the left D-module M. Using module theory, we can then classify the properties of the left D-module M.

4. Homological algebra permits to check these properties of the left D-module M using *extension* and *torsion functors* [42].

5. Gröbner bases over Ore algebras allow to develop effective algorithms which check the properties of the left D-module M, and thus, of the system $R\,z = 0$.

6. Implementations of these algorithms in the package OreModules for the computer algebra system Maple.

Finally, we shall give an introduction to the package OreModules [8] for Maple which offers symbolic methods to investigate the structural properties of multidimensional linear systems over Ore algebras. The advantage of describing these properties in the language of homological algebra carries over to the implementation of OreModules: up to the choice of the domain of operators which occur in a given system, all algorithms are stated and implemented in sufficient generality such that ODEs, PDEs, differential time-delay systems, discrete systems with constant, polynomial or rational coefficients... are covered at the same time.

References

[1] Assan, J. (1999) Analyse et synthèse de l'approche géométrique pour les systèmes linéaires sur un anneau, PhD thesis. Ecole Centrale de Nantes (France).

[2] Becker, T., Weispfenning, V. (1993) Gröbner Bases. A Computational Approach to Commutative Algebra. Springer.

[3] Bender, C. M., Dunne, G. V., Mead, L. R. (2000) Underdetermined systems of partial differential equations. J. Mathematical Physics 41:6388–6398.

[4] Bose, N. K. (1985) Multidimensional Systems Theory: Progess, Directions, and Open Problems. D. Reidel Publishing Company.

[5] Chyzak, F. (1998) Fonctions holonomes en calcul formel. PhD thesis. Ecole Polytechnique (France).

[6] Chyzak, F. *Mgfun* Project. `http://algo.inria.fr/chyzak/mgfun.html`.

[7] Chyzak, F., Salvy, B. (1998) Non-commutative elimination in Ore algebras proves multivariate identities. J. Symbolic Computation 26:187–227.

[8] Chyzak, F., Quadrat, A., Robertz, D. (2002). OreModules project. `http://wwwb.math.rwth-aachen.de/OreModules/`.

[9] Chyzak, F., Quadrat, A., Robertz, D. (2003) Linear control systems over Ore algebras: Effective algorithms for the computation of parametrizations. CDRom of the *Workshop on Time-Delay Systems* (TDS03), IFAC Workshop, INRIA Rocquencourt (France) (08-10/09/03).

[10] Chyzak, F., Quadrat, A., Robertz, D. (2004) OreModules: A symbolic package for the study of multidimensional linear systems, proceedings of the MTNS04, Leuven (Belgique) (05-09/07/04).

[11] Chyzak, F., Quadrat, A., Robertz, D. (2004) Effective algorithms for parametrizing linear control systems over Ore algebras. INRIA Report 5181, available at `http://www.inria.fr/rrrt/rr-5181.html`, to appear in Applicable Algebra in Engineering, Communication and Computing (AAECC).

[12] Conte, G., Perdon, A. M. (2000) Systems over rings: geometric theory and applications, Annual Reviews in Control 24:113–124.

[13] Fliess, M. (1991) Controllability revisited, in Mathematical System Theory. The influence of R. E. Kalman, A. C. Antoulas ed., Springer.

[14] Fliess, M., Mounier, H. (1998) Controllability and observability of linear delay systems: an algebraic approach. ESAIM COCV 3:301–314.

[15] Galkowski, K., Wood, J. ed. (2001) Multidimensional Signals, Circuits and Systems, Taylor and Francis.

[16] Greuel, G.-M., Pfister, G. (2002) A Singular Introduction to Commutative Algebra, Springer.

[17] Habets, L. (1994) Algebraic and computational aspects of time-delay systems. PhD thesis. University of Eindhoven (The Netherlands).

[18] Habets, L. (1996) Computational aspects of systems over rings − reactability and stabilizability, CWI Quarterly 9:85–95.

[19] Kailath, T. (1980) Linear Systems, Prentice-Hall.

[20] Kalman, R. E., Falb, P. L., Arbib, M. A. (1969) Topics in Mathematical Systems Theory, McGraw-Hill.

[21] Kučera, V. (1979) Discrete Linear Control: The Polynomial Equation Approach, Wiley.

[22] Li, H. (2002) Non-commutative Gröbner Bases and Filtered-Graded Transfer, Lecture Notes in Mathematics 1795, Springer.

[23] Malgrange, B. (1963) Systèmes à coefficients constants, Séminaire Bourbaki 1962/63, 246:1–11.

[24] McConnell, J. C., Robson, J. C. (2000) Noncommutative Noetherian Rings. American Mathematical Society.

[25] Mounier, H. (1995) Propriétés structurelles des systèmes linéaires à retards: aspects théoriques et pratiques. PhD Thesis. University of Orsay (France).

[26] Mounier, H., Rudolph, J., Fliess, M., Rouchon, P. (1998) Tracking control of a vibrating string with an interior mass viewed as delay system. ESAIM COCV 3:315–321.

[27] Oberst, U. (1990) Multidimensional constant linear systems. Acta Appl. Math. 20:1–175.

[28] Pillai, H. K., Shankar, S. (1998) A behavioral approach to control of distributed systems. SIAM J. Control and Optimization 37:388–408.

[29] Polderman, J. W., Willems, J. C. (1998) Introduction to Mathematical Systems Theory. A Behavioral Approach. TAM 26, Springer.

[30] Pommaret, J.-F. (2001) Partial Differential Control Theory. Kluwer.

[31] Pommaret, J.-F., Quadrat, A. (1998) Generalized Bezout Identity, Applicable Algebra in Engineering, Communications and Computing 9:91–116.

[32] Pommaret, J.-F., Quadrat, A. (1999) Localization and parametrization of linear multidimensional control systems. Systems and Control Letters 37:247–260.

[33] Pommaret, J.-F., Quadrat, A. (1999) Algebraic analysis of linear multidimensional control systems. IMA J. Control and Optimization 16:275–297.

[34] Pommaret, J.-F., Quadrat, A. (2000) Equivalences of linear control systems, proceedings of MTNS 2000, Perpignan (France), CDRom.

[35] Pommaret, J.-F., Quadrat, A. (2003) A functorial approach to the behaviour of multidimensional control systems, Applied Mathematics and Computer Science 13:7–13.

[36] Pommaret, J.-F., Quadrat, A. (2004) A differential operator approach to multidimensional optimal control, International Journal of Control 77:821–836.

[37] Quadrat, A. (1999) Analyse algébrique des systèmes de contrôle linéaires multidimensionnels. PhD thesis. Ecole Nationale des Ponts et Chaussées (France).

[38] Quadrat, A., Robertz, D. (2005) Parametrizing all solutions of uncontrollable multidimensional linear systems. Proceedings of the 16^{th} IFAC World Congress, Prague (04-08/07/05).

[39] Quadrat, A., Robertz, D. (2005) On the blow-up of stably-free behaviours, to appear in the proceedings of CDC-ECC05, Seville (12-15/12/05).

[40] Sontag, E. (1976) Linear systems over commutative rings: a survey, Ricerche Automat. 7:1–34.

[41] Rosenbrock, H. H. (1970) State Space and Multivariable Theory, J. Wiley.

[42] Rotman, J. J. (1979) An Introduction to Homological Algebra. Academic Press.

[43] Wonham, M. (1985) Linear Multivariable Control: a Geometric Approach, Springer.

[44] Wood, J. (2000) Modules and behaviours in nD systems theory. Multidimensional Dimensional Systems and Signal Processing 11:11–48.

[45] Youla, D. C., Gnavi, G. (1979) Notes on n-dimensional system theory, IEEE Trans. Circuits and Systems 26:259-294.

[46] Zerz, E. (2000) Topics in Multidimensional Linear Systems Theory. Lecture Notes in Control and Information Sciences 256, Springer, London.

Symbolic Analysis of Control Systems

Ruth Hardy
School of Computer Science
University of St Andrews

August 2005

Abstract

This paper introduces a method for formal and symbolic analysis of single–input single–output continuous–time control systems. The method is based on traditional control engineering analysis using Nichols plots and thus focuses on the properties of gain (amplitude) and phase–shift. We reduce Nichols plot requirements to a decision problem and present a procedure to decide problems of this type. An implementation of this procedure requires efficient symbolic manipulation and validated numerical calculation. A prototype tool has been developed using the computer algebra system Maple, the formal theorem prover PVS and the QEPCAD tool for quantifier elimination.

1 Introduction

Control systems are used to augment modern man–made dynamical systems, such as aeroplanes, cars and CD–players. The process for designing control systems is well established and well documented [5], [16], [17]. In traditional control engineering mathematical formulae modelling the behaviour of dynamical and control systems are developed, often as difference or differential equations. These models are considered specifications for systems, which may later be implemented in software or hardware. The models are analysed with respect to some design criteria to ensure they display the correct behaviour before implementation is performed. This *design verification* traditionally uses numeric techniques, often involving the visual inspection of a number of plots, or the performance of a number of simulations. Control systems are becoming more complex and are being relied upon more heavily in safety and mission critical situations during which it is vital to maintain various properties, such as stability, fast response time, or efficient energy usage. It is now widely accepted that traditional informal design verification techniques are no longer sufficient to establish the correctness of the behaviour of systems.

Research into integrating formal or symbolic methods into the development of control systems tends to fall into one of two categories: formal or symbolic methods integrated into classical system development and analysis; (in)finite state or hybrid systems and model checking. Quantifier elimination has been used in the stability analysis of difference schemes in [14]. Various algebraic techniques are used in the traditional analysis of control systems in [10]. A Hoare logic for single–input single–output continuous–time control systems, which is

used to infer properties of the gain and phase–shift of a system based on properties of its subsystems is presented in [6]. An algorithmic approach to the specification and verification of systems as hybrid automata is presented in [2]. A series of abstractions for hybrid automata resulting in discrete transition systems that can be analysed using traditional model checking tools is presented in [19]. The UCLID [7] verification tool for infinite state systems was developed for the analysis of models of both hardware and software systems. The Vapor [4] tool was developed to eliminate the errors that can be introduced by manually translating discrete models into finite state system representations by automatically abstracting models described using behavioural RTL Verilog into the CLU language used by the UCLID tool.

The work described in this paper focuses on integrating formal analysis techniques, including symbolic reasoning, higher order theorem proving, and quantifier elimination, into classical control system analysis, in particular, Nichols plot analysis. In Section 2 Nichols plot analysis is introduced. Section 3 formulates Nichols plot requirements as a decision problem and introduces a set of conditions that can be used in formal and symbolic Nichols plot analysis. Section 4 discusses the need for efficient symbolic computation and validated numeric computation in the automation of formal and symbolic Nichols plot analysis and presents a prototype implementation in the Maple–PVS–QEPCAD system. The work described in this paper is presented in more detail in a forthcoming Calculemus workshop and will be published in the proceedings as a volume of the ENTCS [11].

2 Traditional Nichols Plot Analysis

Nichols plots are a classical method for analysing systems in the frequency domain. A continuous system $F(j\omega)$, where F is a Laplace transform, j is $\sqrt{-1}$ and ω is (non–negative real–valued) frequency, is exposed to a sinusoidal input with varying frequency.[1] The result produced is a sinusoid. A Nichols plot is a parametric plot of the gain (the factor by which the amplitude of the output sinusoid increases the amplitude of the input sinusoid) in decibels $20\log_{10}(|F(j\omega)|)$ against the change in phase $\arctan(\frac{Im(F(j\omega))}{Re(F(j\omega))}) \pm k\pi$. The control system is considered to meet its requirements if the parametric plot stays within or out with a specified region on the graph (for example see Figure 1). This is confirmed visually by the analyst.

The region to be avoided (the *exclusion region*) or conversely the region in which the curve must remain (the *desired region*) depends on the specific requirements for the system, though it can often be specified in terms of linear functions of x that bound it in particular intervals. In their simplest form, these requirements can be reduced to ensuring that in a given interval $[a, b]$ the curve determined by the set of parametric equations $y = Y(\omega)$ and $x = X(\omega)$ lies above or below the bounds $l_i(x)$ of the exclusion/desired region(s). The Nichols plot can be considered an informal and graphical method for solving problems of this type, which provides no guarantee of correct results.

[1] Control engineers conventionally use j rather than i to represent the complex constant and that practice is followed here

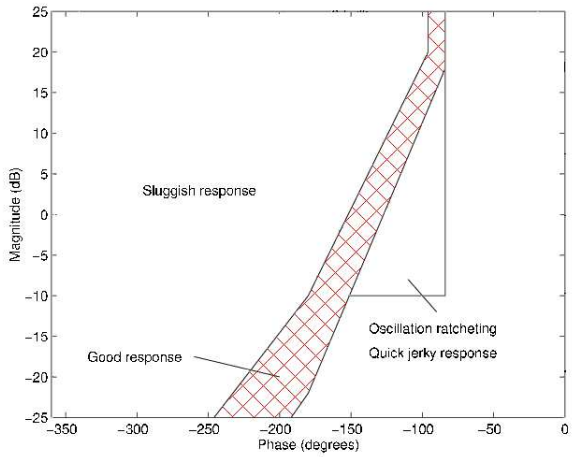


Figure 1: Nichols plot showing an example [9] of a desired region (shaded).

3 Formal and Symbolic Nichols Plot Analysis

In order to perform formal or symbolic analysis of Nichols plot requirements, the requirements must be expressed in some formal manner. Nichols plot requirements can be expressed as conditions on the curve determined by the set of parametric equations for gain $y = Y(\omega)$ and phase–shift $x = X(\omega)$ on a number of disjoint intervals $[a_i, b_i]$. These conditions state that the curve must lie within a number of regions in appropriate intervals. Each of these region can be described as a conjunction of inequalities on the bounds l_i of the exclusion/desired regions. In the intervals $[a_i, b_i]$ the curve may lie in one or more regions. Thus, in each interval the conditions can be organised as a disjunction of conjunctions.

Many procedures exist to decide whether conjunctions and disjunctions of (in)equalities hold. Most of these procedures apply only to real closed fields [8] or to very specific problems involving trigonometric [3] or transcendental functions [20]. Existing procedures do not provide appropriate support to decide problems involving arctan, the natural logarithm or logarithm to the base ten. When the inputs to these three functions are rational functions their derivatives are rational functions (considering ln(10) as a bounded constant) and procedures for real closed fields can easily be applied to (in)equalities involving these derivatives.

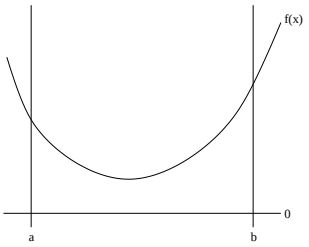
A set of conditions allowing one to reason about a curve based on the derivative and second derivative of the function that defines it have been developed. These conditions examine the convexity of a function[2] along with the value of the function at a number of carefully determined points in a closed interval.

[2]A twice differentiable function $f : \mathbb{R} \to \mathbb{R}$ is convex in intervals in which its second derivative is non–negative (for example x^2) and concave in intervals in which the second derivative is non–positive. A curve $\{(x, y) : y = f(x)\}$ defined by a function $f : \mathbb{R} \to \mathbb{R}$ is *convex/concave* in the intervals in which the function is convex/concave.

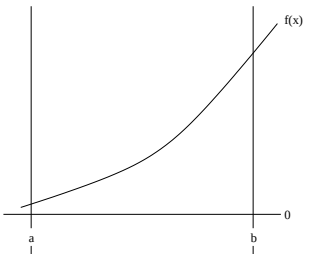
The conditions for convex curves are detailed below and are illustrated for a twice differentiable function $f : \mathbb{R} \to \mathbb{R}$ in the interval $D = [a, b]$. Only the cases for a convex curve $f(x)$ in a closed interval are detailed as concave cases are symmetric to this and can be omitted by looking at $-f(x)$ in appropriate intervals.

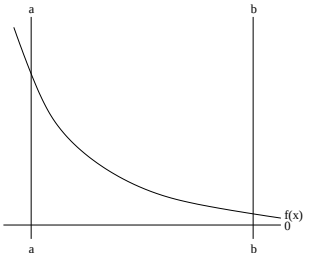
1. The curve is positive on $x \in D$ if and only if one of the following two mutually exclusive conditions holds:

   (a) the gradient of the curve is equal to zero at some point within the interval and the curve is positive at that point, i.e, $\exists x.\, f'(x) = 0 \wedge f(x) > 0$

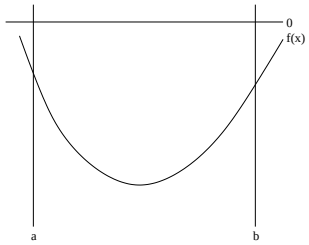


   (b) the gradient of the curve does not equal zero at any point within D and the curve is positive at the limits of the interval, i.e, $\forall x.\, f'(x) \neq 0$, $f(a) > 0$, and $f(b) > 0$





2. The curve is negative on $x \in D$ if and only if the curve is negative at the limits of D, i.e, $f(a) < 0$, and $f(b) < 0$



If none of the above conditions hold for a convex curve then there is at least one point within the interval at which the function defining the curve is equal to zero.

For functions with a finite number of intervals of convexity or concavity (*finitely inflective functions*) any interval of interest can be split into a finite number of intervals over which the curve is either convex or concave and the conditions can be applied in each of these intervals.

4 A Prototype Tool

In order to use the conditions described in Section 3 one must be able to reliably calculate the points of inflection of functions, the convexity of the corresponding curve and the sign of the function at given points. This requires not only powerful symbolic manipulation whose results are guaranteed correct but also validated numerical calculation.

Computer algebra systems (CASs) provide a powerful method for symbolic manipulation and analysis of mathematical formulae and are ideal for performing the transformations and calculations required. However, they can not always guarantee correct results, often ignoring assumptions and side conditions and producing floating point errors during numerical calculation. Formal theorem provers provide powerful methods for formal analysis but lack the ability to perform symbolic manipulation or numerical calculations efficiently. The Maple–PVS [1] tool provides a link between the CAS Maple [15] and the theorem prover PVS [13]. This system allows the calculations performed by Maple to be formally verified by PVS, providing efficient and reliable mathematics. The onus is on Maple to formulate the lemmas to be proved and pass them to PVS along with the proof steps to be taken, usually by invoking some high level PVS strategies.

Decision procedures and quantifier elimination algorithms for real closed fields provide a powerful and efficient method for simplifying or solving lemmas involving polynomials. The QEPCAD–PVS [18] tool allows QEPCAD [12] quantifier elimination routines to be accessed by PVS in the form of PVS strategies. These routines can be used within PVS proofs and the results are considered reliable by PVS.

The Maple–PVS system has been extended to allow the automatic loading of the QEPCAD shared object file into PVS (see Figure 2). This allows Maple to access QEPCAD routines via PVS.

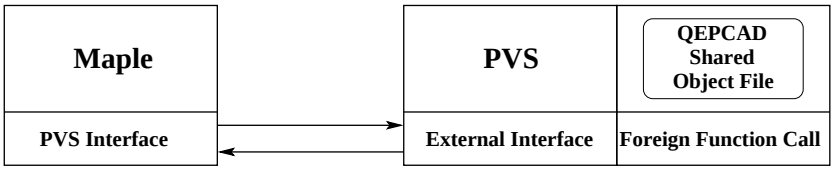


Figure 2: Maple–PVS–QEPCAD.

The prototype tool is implemented in the Maple–PVS–QEPCAD system. The front end of the prototype tool is provided by Maple via Maplets. These provide a Java applet–like graphical user interface into which the input is entered and any results or error messages are displayed. The input is a system represented via the Laplace transform along with a list representing the exclusion/desired region(s) in each of the intervals of interest in terms of a disjunction of conjunctions of inequalities on the bounds of the region. A simple type check mechanism ensures that the input is of the correct type and format. Maple processes the input to form the appropriate lemmas for the application of the conditions of Section 3 and invokes PVS, which in turn may invoke QEPCAD, to perform the required verification. Once the process is complete Maple dis-

plays the results of the analysis and a plot showing the bounding lines for the specified region along with the plot of the curve.

The following describes the steps taken by the prototype tool when considering a system $F(s)$ and an exclusion/desired region, described in terms of a number of disjoint intervals $[a_i, b_i]$, in which there are a number of disjoint regions (e.g, of the form $l_{ij}(x) < y < l_{ik}(x)$) described in terms of the lines $l_{in}(x)$ bounding them.

1. The user supplied input is type checked and if it is not of the correct format an error message is produced and the prototype tool halts.

2. Maple calculates the equations for gain $y = Y(\omega)$ and phase–shift $x = X(\omega)$ of the system $F(j\omega)$.

3. Maple functions are used to calculate, rewrite and simplify the first and second derivatives (expressed as functions of ω) of y (with respect to x) given the set of parametric equations $y = Y(\omega)$ and $x = X(\omega)$. To ensure that no important side conditions have been ignored when performing these calculation PVS is called. PVS attempts to confirm that the function is well defined, is twice differentiable and the derivatives are as specified by Maple. This is a relatively simple task for PVS, which uses the custom built libraries and powerful general purpose simplification and rewrite strategies such as *GRIND* to provide the relevant proofs. If PVS fails to provide the required proof then the prototype tool produces an appropriate error message and halts.

4. To calculate whether the curve is convex or concave in the intervals of interest the interval(s) over ω that correspond to $a_i \leq X(\omega) \leq b_i$ must first be calculated. To calculate these intervals Maple uses inbuilt functions to solve equalities and evaluate floating point numbers to find all solutions ω_{ik} to $a_i = X(\omega)$ or $b_i = X(\omega)$. All non–real and non–positive solutions are discarded, the results are sorted into ascending order, then a point between each $X(\omega_{ik})$ is examined to determine the corresponding intervals. Maple does this using numerical calculation and can suffer from the problem of inexact arithmetic caused by floating point error and may also fail to find all solutions.
   Maple must ensure that the 'solutions' found are good approximations to the actual solutions and that good approximations to all solutions with in $[a_i, b_i]$ have been found. Letting D_i represent small intervals around each of Maple's solution then it must be shown that 1) in each D_i there is exactly one solution and 2) each solution lies within one of the intervals D_i. The tool essentially reapplies the steps described here to provide reassurance that these conditions hold. Since neither of these problems involve parametric equations this is much simpler and does not require step 4; this also means that there will not be any uncontrolled recursion. Also, since the intervals of interest are so small in the first problem it is unlikely that there will be any points of inflection to cause the intervals to be split into subinterval as described in steps 5 and 6, which again simplifies this application.
   To compensate for Maple's inexact arithmetic the solutions are adjusted by a small value to give 'safe' bounds for the interval; for example, if Maple

calculates ω_{ik} and $\omega_{i(k+1)}$ such that $[a_i, b_i] \simeq [X(\omega_{ik}), X(\omega_{i(k+1)})]$ then Maple adjust by some δ such that $[a_i, b_i] \subseteq [X(\omega_{ik} - \delta), X(\omega_{i(k+1)} + \delta)]$. Maple calls PVS to verify that the solutions are indeed 'safe'. If PVS fails to provide any of the required proof then the prototype tool produces an appropriate error message and halts.

5. Maple calculates the points of inflection of $Y(\omega) - l_{ij}(X(\omega))$, including any points at which it becomes vertical, in the intervals $[\omega_{ik} - \delta, \omega_{i(k+1)} + \delta]$. This is achieved using numerical methods to find points at which the second derivative of $Y(\omega) - l_{ij}(X(\omega))$ with respect to x is zero and as a consequence is subject to errors due to inexact arithmetic. In an attempt to avoid this problem Maple calculates small intervals $[p_{ikm} - \delta, p_{ikm} + \delta]$ in which these points should lie (referred to as *intervals of inflection*). PVS is called to confirm not only that these intervals contain true points of inflection rather than points of zero curvature between two regions both strictly convex or concave but also that each of these points is the only point of inflection in an interval and that the derivative of $Y(\omega) - l_{ij}(X(\omega))$ does not equal zero in the interval unless it is exactly at the point of inflection. This is a relatively difficult problem for PVS to solve but since the derivative and second derivative of $Y(\omega) - l_{ij}(X(\omega))$ are rational it is ideal for quantifier elimination. PVS uses the QEPCAD–PVS link to invoke the QEPCAD strategies to verify Maple's results. If PVS fails to provide the required proof then the prototype tool produces an appropriate error message and halts.

6. The intervals $[\omega_{ik} - \delta, \omega_{i(k+1)} + \delta]$ are split into $[\omega_{ik} - \delta, p_{ikm} - \delta]$ $[p_{ikm} - \delta, p_{ikm} + \delta]$ $[p_{ikm} + \delta, \omega_{i(k+1)} + \delta]$ over which the curve is either convex or concave, or is an interval of inflection.

7. Maple formulates the lemmas to be solved by PVS in the form $\lambda\omega \in [a_n, b_n]$. $Y(\omega) - l_{ij}(X(\omega))(\omega) \sim_{ij} 0$, where $\sim_{i,j}$ is the inequality sign indicating whether the curve should lie above ($>$) or below ($<$) the line and $[a_n, b_n]$ are the intervals calculated in step 6. PVS is called by Maple to prove these lemmas; either determining whether the desired case from the set of conditions holds or in the case of intervals of inflection determining, when necessary, the sign of $Y(\omega) - l_{ij}(X(\omega))$ at the bounds of the interval (due to the nature of these intervals the maximum and minimum of $Y(\omega) - l_{ij}(X(\omega))$ must lie on the bounds). Whether the curve lies out with/within any given region in any interval is determined by the conjunction of the truth of the appropriate lemmas.

8. The truth of whether the system meets its Nichols plot requirements in each interval is built up from the disjunction of the truth values for the curve remaining out with/within each of the regions within the interval. The truth of whether the system meets its Nichols plot requirements is then built up from the conjunction of the truth values in each interval. The prototype tool produces an appropriate message stating whether the formula is true or false.

9. Maple displays the truth of this formula along with a plot of the lines and the parametric plot of the curve.

PVS uses custom built libraries containing lemmas concerning the differentiability of various functions important in control system analysis, such as arctan, natural logarithm, logarithm to the base 10, arbitrary rational functions and parametric functions, along with high level strategies and external function calls to QEPCAD to provide the proofs required in the prototype tool. These libraries contain definitions of the natural logarithm and arctan as Taylor series, which allows bounds on the value of these functions for any given input to be defined and numerical calculations using these functions to be validated.

5 Conclusions

The method presented in this paper allows the formal and symbolic analysis of control systems in terms of their Nichols plot requirements. The method requires reliable and efficient mathematics and a prototype tool to automating this method takes advantage of the existing Maple–PVS link to provide reliable mathematics and the QEPCAD–PVS link to utilise existing efficient quantifier elimination techniques. The tool provides a reliable and rigorous analysis of systems allowing the user of the tool to have little or no knowledge of formal methods.

6 Acknowledgements

Thanks go to Richard Boulton and Ursula Martin for their help and guidance, especially in the early stages of this work, also to Roy Dyckhoff and Steve Linton for their continuing help and guidance. Additional thanks go to John Hall, Rick Hyde and Yoge Patel for sharing their insights into control engineering, and to Rob Arthan, Tom Kelsey and Colin O'Halloran for many helpful discussions.

References

[1] A Adams, M Dunstan, H Gottliebsen, T Kelsey, U Martin, and S Owre. Computer algebra meets automated theorem proving: Integrating Maple and PVS. In R. J Boulton and P. B Jackson, editors, *Proceedings of the 14th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2001)*, volume 2152 of *Lecture Notes in Computer Science*, pages 27–42. Springer–Verlag, 2001.

[2] R Alur, C Courcoubetis, T. A Henzinger, and P. H Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In R. L Grossman, A Nerode, A. P Ravn, and H Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer, 1993.

[3] H Anai and V Weispfenning. Deciding linear–trigonometric problems. In C Traverso, editor, *ISSAC '00: Proceedings of the 2000 international symposium on Symbolic and algebraic computation*, pages 14–22. ACM Press, 2000.

[4] Z. S Andraus and K. A Sakallah. Automatic abstraction and verification of Verilog models. In S Malik, L Fix, and A. B Kahng, editors, *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 218–223. ACM Press, 2004.

[5] O. H Bosgra, H Kwakernaak, and G Meinsma. *Design Methods for Control Systems: Notes for a course of the Dutch Institute of Systems and Control, Winter term 2001–2002*. Department of Systems, Signals and Control, University of Twente, 2001.

[6] R Boulton, R Hardy, and U Martin. A Hoare logic for single–input single–output continuous–time control systems. In A Pnueli and O Maler, editors, *Proceedings of the 6th International Workshop on Hybrid Systems: Computation and Control (HSCC 2003)*, volume 2623 of *Lecture Notes in Computer Science*, pages 113–125. Springer, 2003.

[7] R. E Bryant, S. K Lahiri, and S. A Seshia. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In E Brinksma and K Guldstrand Larsen, editors, *CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification*, pages 78–92. Springer–Verlag, 2002.

[8] B. F Caviness and J. R Johnson, editors. *Quantifier elimination and cylindrical algebraic decomposition*. Springer Wien NewYork, 1998.

[9] Action Group FM(AG08). Robust flight control design challenge problem formulation and manual: the high incidence research model (HIRM). Technical Report TP–088–4, version 3, Group for Aeronautical Research and Technology in Europe (GARTEUR), 1997.

[10] K Forsman. *Constructive commutative algebra in nonlinear control theory*. PhD thesis, Linköping University, 1991.

[11] R Hardy. Interactions between PVS and Maple in symbolic analysis of control systems. In J Carette and W. M Farmer, editors, *Proceedings of the 12th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus 2005)*. ENTCS, 2005.

[12] H Hong. Qepcad. Available at `http://www.cs.usna.edu/~qepcad/B/QEPCAD.html`.

[13] SRI International. PVS. Available at `http://pvs.csi.sri.com`.

[14] R Liska and S Steinberg. Applying quantifier elimination to stability analysis of difference schemes. *Comput. Journal*, 36(5):497–503, 1993.

[15] M. B Monagan, K. O Geddes, K. M Heal, G Labahn, S. M Vorkoetter, J McCarron, and P DeMarco. *Maple 7 programming Guide*. Waterloo Maple Inc, 2001.

[16] K Ogata. *Modern control engineering*. Prentice–Hall, third edition, 1997.

[17] R. W Pratt, editor. *Flight control systems: Practical issues in design and implementation*, volume 57 of *IEE Control Engineering Series*. The Institution of Electrical Engineers, 2000. Copublished by The American Institute of Aeronautics and Astronautics.

[18] A Tiwari. PVS–QEPCAD. Available at `http://www.csl.sri.com/users/tiwari/qepcad.html`.

[19] A Tiwari and G Khanna. Series of abstractions for hybrid automata. In C. J Tomlin and M. R Greenstreet, editors, *Hybrid Systems: Computation and Control HSCC*, volume 2289 of *LNCS*, pages 465–478. Springer, March 2002.

[20] V Weispfenning. Deciding linear–exponential problems. *SIGSAM Bullettin*, 34(1):30–31, 2000.

Transfer Principle Proof Tactic for Nonstandard Analysis

Brian Huffman*

3rd August 2005

Abstract

This paper describes a type constructor for nonstandard extensions of arbitrary types, generalizing the previous formalization of nonstandard analysis in Isabelle by Fleuriot and Paulson [1]. In addition to numeric types, nonstandard extensions of function spaces and set types also prove useful—they provide an elegant way to define internal functions and internal sets.

The paper also describes an implementation of the transfer principle of nonstandard analysis as an Isabelle proof tactic. When applied to an appropriate subgoal, the transfer tactic replaces the subgoal with a logically equivalent one that does not refer to nonstandard types. The tactic produces a proof of the equivalence; it does not generate any new axioms.

1 Introduction

Nonstandard analysis [2] was developed in the 1960s by Abraham Robinson to provide a rigorous foundation for the use of infinitesimal numbers in mathematics. Nonstandard analysis introduces an extension of the reals called the hyperreals, which includes infinitesimal and infinite numbers in addition to the standard reals. The hyperreals also preserve many properties of the reals, according to the transfer principle: Any true first order statement involving arithmetic on the reals may be reinterpreted as a true first order statement about the hyperreals.

Most calculus textbooks define derivatives and other notions of calculus using the epsilon-delta definition of limits. The hyperreal numbers require more work to formalize, but they offer significant advantages. First, using infinitesimals is often more intuitive: For example, in nonstandard analysis we can interpret the derivative $\frac{dy}{dx}$ as an actual ratio of two infinitesimal numbers. Second, by avoiding epsilon-delta limits, nonstandard analysis reduces the number of quantifiers we must deal with in proofs. This is especially important for automation in a theorem prover, where quantifiers are notoriously difficult to handle.

*OGI School of Science and Engineering at OHSU, Beaverton, OR 97006

1.1 Isabelle/HOL

Isabelle is a generic interactive theorem prover, which can be instantiated with various kinds of object-logics. Isabelle/HOL is an instantiation of higher order logic.

The formula syntax in Isabelle/HOL includes standard logical notation for connectives, quantifiers, and set operations. In addition, Isabelle has separate syntax for the meta-level logic: $\bigwedge$, $\Longrightarrow$, and $\equiv$ represent meta-level universal quantification, implication, and equality. There is also notation for nested meta-level implication: $[\![P_1; \ldots; P_n]\!] \Longrightarrow R$ is short for $P_1 \Longrightarrow \cdots \Longrightarrow P_n \Longrightarrow R$. Other specialized Isabelle syntax will be introduced as it is used.

The syntax of types is similar to the language ML, except that Isabelle uses a double arrow ($\Rightarrow$) for function types. Some binary type constructors are written infix, as in the product type $nat \times bool$; other type constructors are written postfix, as in *bool list* or *nat set*. Finally, $'a$ and $'b$ denote free type variables.

Isabelle theories declare new constants with the **consts** keyword. Definitions may be supplied later using **defs**; alternatively, constants may be declared and defined at once using **constdefs**. Theories introduce new types with the **typedef** command, which defines a type isomorphic to a given non-empty set. The keywords **lemma** and **theorem** introduce theorems.

1.2 HOL-Complex

Fleuriot and Paulson [1] have already developed a theory of nonstandard analysis in Isabelle/HOL: Their development comprises the HOL-Complex theory, which is part of the Isabelle distribution. The HOL-Complex theory has several parts. First, it includes a formalization of a few ordinary numeric types, including the rationals and the reals. Next, there is a formalization of free ultrafilters, which are used to define nonstandard types. Three separate nonstandard types are defined, each with their own specific set of operations: hyperreals, hypernaturals, and the hypercomplex numbers. Finally, these nonstandard types are used to formalize various concepts in real analysis.

The development described in the remainder of this paper may be considered as a potential replacement for the middle part of the HOL-Complex theory. It reuses much of the first part of HOL-Complex, including the formalization of standard numeric types and free ultrafilters. The remainder of the development consists of a framework for defining nonstandard types, defining operations on nonstandard types, and proving properties about them.

2 Type Constructor for Nonstandard Analysis

The theory starts with the definition of the *star* type constructor. The type $'a$ *star* is defined as the set of equivalence classes of the *starrel* relation. (Note that double-slash // is Isabelle syntax for the set quotient operator.) In turn, *starrel* is defined in terms of an arbitrary free ultrafilter over the natural numbers, using Hilbert's indefinite choice operator. The development of nonstandard analysis is non-constructive, with the existence of a free ultrafilter proven using the axiom of choice. (See Appendix A for a review of free ultrafilters.)

constdefs
FreeUltrafilterNat :: *nat set set* (*𝒰*)
𝒰 ≡ *SOME U*. *FreeUltrafilter U*

constdefs
starrel :: ((*nat* ⇒ *'a*) × (*nat* ⇒ *'a*)) *set*
starrel ≡ {(*X*, *Y*). {*n*. *X n* = *Y n*} ∈ *𝒰*}

typedef *'a star* = (*UNIV* :: (*nat* ⇒ *'a*) *set*) // *starrel*

The **typedef** command generates functions *Rep-star* and *Abs-star* to convert between the new type *'a star* and the representation type (*nat* ⇒ *'a*) *set*. Given a value of type *'a star*, *Rep-star* returns its equivalence class of sequences; *Abs-star* is the inverse mapping. The result is undefined when *Abs-star* is applied to a set that is not an equivalence class.

Abs-star is not very convenient to use by itself, so we now define some other basic functions to construct values of type *'a star*. The function *star-n* returns the value corresponding to the equivalence class of a given sequence. It is not injective, but it is surjective, which means that *star-n* may be used for doing case analysis on values of type *'a star*. The function *star-of* is an injective function that returns "standard" values of type *'a star*, which correspond to constant sequences.

constdefs
star-n :: (*nat* ⇒ *'a*) ⇒ *'a star*
star-n X ≡ *Abs-star* {*Y*. (*X*, *Y*) ∈ *starrel*}

star-of :: *'a* ⇒ *'a star*
star-of x ≡ *star-n* (*λn*. *x*)

The HOL-Complex theory defines lifted versions of many functions on real numbers. For example, addition on reals (of type *real* ⇒ *real* ⇒ *real*) is lifted to addition on hyperreals (of type *hypreal* ⇒ *hypreal* ⇒ *hypreal*). The HOL-Complex theory defines a large number of these lifted functions, all in a similar way.

To generalize over this basic pattern of definition, we now define a function *Ifun* that is essentially a lifted version of the two-argument application function (*λf x*. *f x*). It is called *Ifun* because its range is exactly the set of internal functions. It is similar to the function *starfun-n* from HOL-Complex, but instead of taking a sequence of functions as an argument, it takes a value of type (*'a* ⇒ *'b*) *star*, which represents an equivalence class of such sequences.

constdefs
Ifun :: (*'a* ⇒ *'b*) *star* ⇒ *'a star* ⇒ *'b star* (*infixl* ⋆ *300*)
Ifun f ≡ *λx*. *Abs-star*
(⋃ *F*∈*Rep-star f*. ⋃ *X*∈*Rep-star x*. {*Y*. (*λn*. *F n* (*X n*), *Y*) ∈ *starrel*})

lemma *Ifun-star-n*: *star-n F* ⋆ *star-n X* = *star-n* (*λn*. *F n* (*X n*))

lemma *Ifun* [*simp*]: *star-of f* ⋆ *star-of x* = *star-of* (*f x*)

Using *star-of* and *Ifun* as basic combinators, it is possible to define lifted versions of functions of any arity. It takes some non-trivial reasoning using the

definition of *starrel* to establish the basic properties of *Ifun*, but we only have to do this once: For any function defined in terms of *Ifun*, we can then easily derive its properties from the *Ifun* lemmas.

One more useful combinator function is *unstar*, which converts from type *bool star* to type *bool*. It comes in very handy for defining lifted versions of predicates.

constdefs
unstar :: *bool star* ⇒ *bool*
unstar b ≡ *b* = *star-of True*

lemma *unstar-star-n*: *unstar* (*star-n P*) = ({*n*. *P n*} ∈ $\mathcal{U}$)

lemma *unstar* [*simp*]: *unstar* (*star-of p*) = *p*

Next we use *star-of*, *Ifun*, and *unstar* to define several useful functions for defining lifted functions and predicates.

constdefs
Ifun-of :: ('*a* ⇒ '*b*) ⇒ ('*a star* ⇒ '*b star*)
Ifun-of f ≡ *Ifun* (*star-of f*)

Ifun2 :: ('*a* ⇒ '*b* ⇒ '*c*) *star* ⇒ ('*a star* ⇒ '*b star* ⇒ '*c star*)
Ifun2 f ≡ λ*x y*. *f* ⋆ *x* ⋆ *y*

Ifun2-of :: ('*a* ⇒ '*b* ⇒ '*c*) ⇒ ('*a star* ⇒ '*b star* ⇒ '*c star*)
Ifun2-of f ≡ λ*x y*. *star-of f* ⋆ *x* ⋆ *y*

Ipred :: ('*a* ⇒ *bool*) *star* ⇒ ('*a star* ⇒ *bool*)
Ipred P ≡ λ*x*. *unstar* (*P* ⋆ *x*)

Ipred-of :: ('*a* ⇒ *bool*) ⇒ ('*a star* ⇒ *bool*)
Ipred-of P ≡ λ*x*. *unstar* (*star-of P* ⋆ *x*)

Ipred2 :: ('*a* ⇒ '*b* ⇒ *bool*) *star* ⇒ ('*a star* ⇒ '*b star* ⇒ *bool*)
Ipred2 P ≡ λ*x y*. *unstar* (*P* ⋆ *x* ⋆ *y*)

Ipred2-of :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ ('*a star* ⇒ '*b star* ⇒ *bool*)
Ipred2-of P ≡ λ*x y*. *unstar* (*star-of P* ⋆ *x* ⋆ *y*)

We can also use *star-of*, *Ifun*, and *unstar* to define a function that produces internal sets. It is similar to the function *starset-n* from HOL-Complex, but like *Ifun* it takes a value of type *'a set star* as an argument instead of a sequence.

constdefs
Iset :: '*a set star* ⇒ '*a star set*
Iset A ≡ {*x*. *Ipred2-of* (*op* ∈) *x A*}

Iset-of :: '*a set* ⇒ '*a star set*
Iset-of A ≡ *Iset* (*star-of A*)

lemma *Iset-star-n*:
(*star-n X* ∈ *Iset* (*star-n A*)) = ({*n*. *X n* ∈ *A n*} ∈ $\mathcal{U}$)

Finally, we can define many standard overloaded constants for the *star* type constructor, using the lifting functions defined above.

defs (**overloaded**)
star-zero-def: $0 \equiv \mathit{star\text{-}of}\ 0$
star-one-def: $1 \equiv \mathit{star\text{-}of}\ 1$
star-number-def: $\mathit{number\text{-}of}\ b \equiv \mathit{star\text{-}of}\ (\mathit{number\text{-}of}\ b)$
star-add-def: $(op\ +) \equiv \mathit{Ifun2\text{-}of}\ (op\ +)$
star-diff-def: $(op\ -) \equiv \mathit{Ifun2\text{-}of}\ (op\ -)$
star-minus-def: $\mathit{uminus} \equiv \mathit{Ifun\text{-}of}\ \mathit{uminus}$
star-mult-def: $(op\ *) \equiv \mathit{Ifun2\text{-}of}\ (op\ *)$
star-divide-def: $(op\ /) \equiv \mathit{Ifun2\text{-}of}\ (op\ /)$
star-inverse-def: $\mathit{inverse} \equiv \mathit{Ifun\text{-}of}\ \mathit{inverse}$
star-le-def: $(op\ \leq) \equiv \mathit{Ipred2\text{-}of}\ (op\ \leq)$
star-less-def: $(op\ <) \equiv \mathit{Ipred2\text{-}of}\ (op\ <)$
star-abs-def: $\mathit{abs} \equiv \mathit{Ifun\text{-}of}\ \mathit{abs}$
star-div-def: $(op\ \mathit{div}) \equiv \mathit{Ifun2\text{-}of}\ (op\ \mathit{div})$
star-mod-def: $(op\ \mathit{mod}) \equiv \mathit{Ifun2\text{-}of}\ (op\ \mathit{mod})$
star-power-def: $(op\ \hat{}) \equiv \lambda x\ n.\ \mathit{Ifun\text{-}of}\ (\lambda x.\ x\ \hat{}\ n)\ x$

3 Transfer Tactic

The transfer principle is a meta-mathematical theorem, which says that many propositions over nonstandard types are logically equivalent to syntactically similar propositions over standard types. Such a principle cannot be encoded as a single theorem in Isabelle because it quantifies over valid theorems, which are not first class entities in Isabelle's object logic. Instead, we can encode it as an algorithm that can generate any of an entire family of theorems.

In this development, the transfer principle is implemented as a proof tactic. When applied to an appropriate subgoal, it replaces the subgoal with a logically equivalent one that does not refer to the *star* type constructor. The tactic produces a proof of the equivalence; it does not generate any new axioms.

3.1 Stating the Equivalence

When the transfer tactic is first called, it obtains a syntactic representation of the current subgoal as an ML term datatype. The term is expected to represent a proposition about *star* types. The job of the tactic is to replace the subgoal with an equivalent one that does not mention *star* types.

Therefore, the first thing that the tactic does is to traverse the term, removing all references to the *star* type constructor from the types of constants, and also completely removing any constants like *Ifun*, *star-of*, *Iset*, etc. Hopefully the resulting term will still be type-correct: The types of constants like *Ifun* and *star-of* reduce to the type of an identity function when un-starred, so they can safely be removed. Any other constant whose type mentions *star* should be polymorphic, so that it will still work at the un-starred instance. For example, the equality operator is valid at both type $'a\ \mathit{star} \Rightarrow 'a\ \mathit{star} \Rightarrow \mathit{bool}$ and type $'a \Rightarrow 'a \Rightarrow \mathit{bool}$.

Next, the tactic creates a new term that states an equivalence between the original term and its un-starred version. If this equivalence term type checks, then the tactic will attempt to prove the equivalence.

3.2 Starting the Equivalence Proof

We start with an equivalence between terms of type *prop*, which is the type of truth values in Isabelle's meta-logic. The first step is to reduce this to an equivalence between terms of type *bool*, the type of truth values in the object logic. Therefore we unfold a set of rewrite rules that convert all of the meta-level quantification, implication, and equalities to ordinary object-level constructs.

The next step involves unfolding a few different sets of definitions. First of all, the tactic unfolds all of the definitions for the overloaded constants listed in the previous section, including *star-zero-def*, *star-add-def*, *star-le-def*, etc. The tactic also unfolds the definitions of constants like *Ifun-of*, *Ipred*, and *star-of*. After all this unfolding, the remaining equivalence subgoal should only mention the constants *Ifun*, *Iset*, *star-n*, boolean operators, quantifiers, and a few other constants that the transfer principle knows about.

Once the equivalence has been reduced to a manageable form, the following introduction rule starts the remainder of the proof. (*Trueprop* is the function in Isabelle that maps from type *bool* to type *prop*—it is usually implicit.)

lemma *transfer-start*:
$P \equiv \{n.\ Q\} \in \mathcal{U} \Longrightarrow \mathit{Trueprop}\ P \equiv \mathit{Trueprop}\ Q$

3.3 Transfer Introduction Rules

The remaining steps of the proof are completely syntax-directed. At each step, the top-level connective determines an appropriate introduction rule to apply. Each argument to the top-level connective generates a new subgoal, each of which also takes the form of an equivalence.

The transfer introduction rules for unary negation, conjunction, and existential quantification over *star* types are all proven using the properties of free ultrafilters. Similar introduction rules for other boolean operators and quantifiers can be derived from these rules.

lemma *transfer-not*:
$[\![p \equiv \{n.\ P\ n\} \in \mathcal{U}]\!] \Longrightarrow \neg\ p \equiv \{n.\ \neg\ P\ n\} \in \mathcal{U}$

lemma *transfer-conj*:
$[\![p \equiv \{n.\ P\ n\} \in \mathcal{U};\ q \equiv \{n.\ Q\ n\} \in \mathcal{U}]\!]$
$\Longrightarrow p \wedge q \equiv \{n.\ P\ n \wedge Q\ n\} \in \mathcal{U}$

lemma *transfer-ex*:
$[\![\bigwedge X.\ p\ (\mathit{star\text{-}n}\ X) \equiv \{n.\ P\ n\ (X\ n)\} \in \mathcal{U}]\!]$
$\Longrightarrow \exists x{::}'a\ \mathit{star}.\ p\ x \equiv \{n.\ \exists x.\ P\ n\ x\} \in \mathcal{U}$

The above introduction rules only deal with equivalences between booleans. However, it is usually the case that some subterms will have non-boolean types, for example *star* types or sets. Rules for constants with these other types are similar, but have a different form on the right hand side: For *star* types the right side contains an application of *star-n*, and for sets the right hand side starts with *Iset*.

lemma *transfer-eq*:
$[\![x \equiv \mathit{star\text{-}n}\ X;\ y \equiv \mathit{star\text{-}n}\ Y]\!] \Longrightarrow x = y \equiv \{n.\ X\ n = Y\ n\} \in \mathcal{U}$

lemma *transfer-if*:
$[\![p \equiv \{n.\ P\ n\} \in \mathcal{U};\ x \equiv \mathit{star\text{-}n}\ X;\ y \equiv \mathit{star\text{-}n}\ Y]\!]$
$\Longrightarrow (\mathit{if}\ p\ \mathit{then}\ x\ \mathit{else}\ y) \equiv \mathit{star\text{-}n}\ (\lambda n.\ \mathit{if}\ P\ n\ \mathit{then}\ X\ n\ \mathit{else}\ Y\ n)$

lemma *transfer-mem*:
$[\![x \equiv \mathit{star\text{-}n}\ X;\ a \equiv \mathit{Iset}\ (\mathit{star\text{-}n}\ A)]\!]$
$\Longrightarrow x \in a \equiv \{n.\ X\ n \in A\ n\} \in \mathcal{U}$

lemma *transfer-set-eq*:
$[\![a \equiv \mathit{Iset}\ (\mathit{star\text{-}n}\ A);\ b \equiv \mathit{Iset}\ (\mathit{star\text{-}n}\ B)]\!]$
$\Longrightarrow a = b \equiv \{n.\ A\ n = B\ n\} \in \mathcal{U}$

Each application of one of these introduction rules results in a smaller subgoal. Eventually, the left hand side of the subgoals will reduce to an atomic term, which can be discharged by one of the following rules.

lemma *transfer-star-n*: $\mathit{star\text{-}n}\ X \equiv \mathit{star\text{-}n}\ (\lambda n.\ X\ n)$

lemma *transfer-bool*: $p \equiv \{n.\ p\} \in \mathcal{U}$

After the equivalence proof is done, the transfer tactic uses the resulting equivalence theorem as a rewrite rule to replace the old subgoal with the new, un-starred version.

3.4 Using the Transfer Tactic

I have used the transfer tactic to convert two large collections of theorems from ordinary types to star types. First, it is easily proved that the *star* type constructor inherits membership in a large number of axiomatic type classes from its argument type; for example, for any ordered field $'a$, $'a\ \mathit{star}$ is also an ordered field. For each class axiom, invoking the transfer tactic reduces the proof obligation at type $'a\ \mathit{star}$ to type $'a$, which may be discharged by applying the class axiom at that type.

One axiomatic type class that presents a slight difficulty is the *recpower* class, whose axioms assert that the exponentiation operator ($\mathit{op}\ \hat{}\ ::\ 'a \Rightarrow \mathit{nat} \Rightarrow 'a$) is defined in a standard way. When we try to prove that $'a\ \mathit{star}$ inherits the *recpower* class from $'a$, we get the following two subgoals:

$\bigwedge(a::'a\ \mathit{star}).\ a\ \hat{}\ 0 = 1$
$\bigwedge(a::'a\ \mathit{star})\ (n::\mathit{nat}).\ a\ \hat{}\ \mathit{Suc}\ n = a * a\ \hat{}\ n$

The transfer tactic handles the first subgoal just fine, because it only uses universal quantification over star types. Indeed, this is the situation for the vast majority of numeric class axioms. However, the second subgoal also quantifies over type *nat*, which the transfer tactic does not handle. The workaround is to prove, as a lemma, a modified version of the second subgoal where only a is universally quantified, and n is a free variable.

The second major use case is the conversion of the entire Isabelle/HOL natural number theory into the *nat star* type. For each theorem in the original theory file, a new version is stated that uses type *nat star* in place of *nat*, and *Ifun-of Suc* in place of *Suc*. It is also necessary to add explicit meta-universal quantifiers for each free variable of type *nat star* in each theorem, because the transfer tactic is not allowed to change the types of free variables in the

middle of a proof. (This does not affect the resulting theorem, because Isabelle automatically discharges meta-universal quantifiers after the end of any proof.) The proof script for each of the theorems is a one-liner: First apply the transfer tactic, and then apply the original rule from the theory of natural numbers.

The only theorems from the natural number theory that require extra modifications are the induction rules, because they involve universal quantification over predicates $P :: nat \Rightarrow bool$. In other presentations of nonstandard analysis, the transfer principle produces induction rules for *nat star* with extra side conditions that restrict $P :: nat\ star \Rightarrow bool$ to be an internal predicate. In this development, however, no side conditions are necessary: The transfer tactic can generate induction rules that quantify over $P :: (nat \Rightarrow bool)\ star$, whose type can represent only internal predicates.

4 Conclusion

The development described in this paper borrows much from the HOL-Complex theory. My definition of a type constructor for nonstandard types is a straightforward generalization of the definitions of nonstandard types in the HOL-Complex theory. I have also reused its formalization of free ultrafilters, and its formalization of the real number system.

One original contribution of this work is to refactor the common patterns of definitions into a few basic combinators, from which many other concepts may be defined. Together with the transfer principle, this allows reasoning about nonstandard types without ever having to look at how they are represented in terms of sequences using ultrafilters. To the end user, it is almost as if nonstandard analysis had been formalized axiomatically.

The primary benefit to this work, however, is the degree of automation that it brings to the formalization of nonstandard analysis. Many theorems that previously required complex proofs can now be done in a straightforward and almost trivial manner.

References

[1] Jacques D. Fleuriot and Lawrence C. Paulson. Mechanizing nonstandard real analysis. In *LMS J. Comput. Math.*, pages 140–190, 2000.

[2] Abraham Robinson. *Non-standard Analysis*. Princeton Landmarks in Mathematics. Princeton University Press, 1996.

A Free Ultrafilters

This section is a quick overview of free ultrafilters and how they are used to define the hyperreal numbers. Let U be a set of sets of naturals, and let the elements of U be called the "large" sets. Here are some properties of large sets that we might like to have:

1. The set of all naturals is large, and the empty set is not.

2. If A and B are large, then so is $A \cap B$.

3. If A is large and $A \subseteq B$, then B is large.

4. If A is not large, then the complement of A is large.

5. If A is finite, then A is not large.

If U satisfies the first three properties, then we say that U is a *filter*. If U additionally satisfies Property 4, then U is an *ultrafilter*; and if U satisfies all five then U is a *free ultrafilter*. Using Zorn's Lemma (an equivalent form of the axiom of choice) it is possible to prove the existence of a free ultrafilter over the natural numbers.

We can use a free ultrafilter to define the hyperreal numbers as equivalence classes of sequences of real numbers. We will say that two sequences X and Y are equivalent ($X \sim Y$) if their agreement set $\{n.\ X(n) = Y(n)\}$ is large. Properties 1–3 ensure that this is an equivalence relation.

We can also use the ultrafilter to define a less-than relation on sequences of reals. We will say that $X < Y$ if the set $\{n.\ X(n) < Y(n)\}$ is large. Properties 1–3 ensure that the less-than relation respects the equivalence relation on sequences. Property 4 ensures that the order trichotomy of reals is preserved: For any sequences X and Y, exactly one of $X < Y$, $Y < X$, or $X \sim Y$ must hold.

Given any real number x, we can define a corresponding "standard" hyperreal as the equivalence class of the constant sequence $(x, x, x, \ldots)$. Property 1 ensures that this is an injective mapping. Property 5 is used to show that this mapping is not surjective. Consider the sequence $(1, 2, 3, \ldots)$ which does not contain more than one occurrence of any number. This sequence is not equivalent to any constant sequence, since the agreement set between the two is finite, and thus not large. In fact, the hyperreal corresponding to the sequence $(1, 2, 3, \ldots)$ is greater than any standard number.

Towards a Hoare logic for abstract systems

Erik Arne Mathiesen
Queen Mary College, University of London

1 Introduction

With the ever-increasing complexity of systems in the fields of engineering and computer science, testing by traditional methods, such as simulation, very often fails to address all functional issues and bugs. This has led to an increasing need for formal verification techniques to determine whether or not systems in question uphold safety-critical properties. Formal approaches to system verification is wide-spread now in safety-critical hardware and software design, although it is virtually unexplored outside these areas.

By an abstract system we mean a framework describing a collection of systems. We propose a framework for reasoning about abstract systems using a logical framework in the style of Hoare logic [3]. This framework can be used to formally prove properties of abstract systems. The approach we take is a direct extension of previous work done on linear control systems [7]. The setting of our abstract systems will be the mathematical language of category theory [1, 5]. Using category theory we will define our notion of abstract systems as well as our notion the computational behavior of such. With this general setting we will show how to extract some concrete logical frameworks for reasoning about traditional systems, in particular systems described by linear differential equations, as instantiations of our abstract framework.

2 System theories and computational theories

We choose to make a distinction between what we call the statics of the systems, described by the system theories, and the dynamics of the systems, described by the computational theories. In other words, the system theory describes the structure of the systems, whereas the computational theory describes how they interact. A system theory can have many different computational theories describing the dynamics of the systems, although ultimately, these different computational theories should be representations of the same behavior.

Formally we define a system theory to be a subcategory of the category **Rel** of relations and sets and view each system of the system theory simply as a morphism in the category, i.e. as a relation between two sets. We define our computational theories to be traced monoidal subcategories [4] of **Rel** and we link the two together with a functor, called a computational interpretation, which maps a system theory to a computational theory and which hits all objects and morphisms in its codomain.

3 A Hoare logic for computational theories

Having defined our quite basic notions of system theories and computational theories we can define a logical framework for reasoning about the computational behavior of a class systems, in other words a logical framework for reasoning about computational theories. Our framework will be Hoare logic-styled in the sense that we will define a notion of Hoare triples for the morphisms of our computational theories.

To be able to define Hoare triples of morphisms we need to be able to talk about restrictions of morphisms to subsets of its source and target, since we are working on the category of relations and sets this is not to difficult to archive, but note that the construction used could quite easily be generalized to arbitrary categories.

In the following let $\mathcal{C}$ be an arbitrary subcategory of **Rel**. If $x \in \mathcal{C}_0$ then we define the powerset mapping $\mathcal{P} : \mathcal{C}_0 \to \mathcal{S}et$ to be

$$\mathcal{P}(x) = \{y \in \mathbf{Rel}_0 \mid y \subseteq x\}$$

If $f : x \to y \in \mathcal{C}_1$ then we define the powerset extension $f^* : \mathcal{P}(x) \to \mathcal{P}(y)$ of f to be

$$f^*(p) = \{a \in y \mid \exists b \in p(bfa)\}$$

for all $p \in \mathcal{P}(x)$. This way we can talk about restrictions of a morphism with regards to it source and target and we can hence define the Hoare triple of morphism similar to the traditional case. Note that if $f : x \to y \in \mathcal{C}_1$ then intuitively the Hoare-triple $\{p\}f\{q\}$ holds whenever $f^*(p) \subseteq q$ for all objects $p \in \mathcal{P}(x)$ and $q \in \mathcal{P}(y)$.

Using the above notion of a Hoare triple we define a logical framework of inference rules with which one can reason about morphisms of any computational theory. The logical framework consists of a set of inference rules, shown below, one for each operation of the computational theories, i.e. composition, monoidal product and trace, and one axiom scheme, axiomatising the intuitive meaning of Hoare triples, as discussed above.

- Composition
$$\frac{\{p\}f\{q\} \quad \{q\}g\{r\}}{\{p\}g \circ f\{r\}} \ (\circ)$$
where $f : x \to y, g : y \to z$ and $p \in \mathcal{P}(x)$, $q \in \mathcal{P}(y)$, $r \in \mathcal{P}(z)$.

- Monoidal product
$$\frac{\{p\}f\{q\} \quad \{r\}g\{s\}}{\{p \otimes r\}f \otimes g\{q \otimes s\}}$$
where $f : x \to y, g : z \to w$ and $p \in \mathcal{P}(x)$, $q \in \mathcal{P}(y)$, $r \in \mathcal{P}(z)$, $s \in \mathcal{P}(w)$.

- Trace
$$\frac{\{p \otimes z\}f\{q \otimes z\}}{\{p\}Tr^z_{x,y}(f)\{q\}}$$
where $f : x \times z \to y \times z$ and $p \in \mathcal{P}(x)$, $q \in \mathcal{P}(y)$.

As mentioned the logical systems has one axiom scheme, as shown below, which is defined for all objects in theory and all morphisms which are in the set I of so-called atomic systems.
We introduce the notion of atomic systems as an abstraction of basic constructs as known from actual systems, e.g. assignments in program languages, registers and scalar-multiplication in stream circuits etc. We define a Hoare computational theory to be a computational theory with a set of atomic systems such that every morphism of the theory can be constructed using a finite series of atomic systems and the basic operations of the theory. Furthermore a Hoare computational theory must for all pairs of objects include a subset inclusion morphism between these, this provides us with weakening and strengthening rules as special cases of our composition rule.

4 Instantiations

Several instantiations of the framework are currently under investigation, these include stream calculus and stream circuits [8], while programs [3], linear control systems [6] and the geometry of interaction [2]. These instantiations of the general framework into actual system theories provides an interesting setting for examining the system theories in question and particularly the behavior of feedbacks in these systems. An interesting result of the instantiations is the observation that the instantiation of the language of while programs actually instantiates as the original Hoare logic. This shows that our framework is indeed a generalisation of Hoare logic in the setting of abstract systems.

Acknowledgments
This research is being done in collaboration with Dr. Paulo Oliva and Prof. Ursula Martin and is being supported by EPSRC grant GR/S31242/01 and carried out as part of the project "Logical Structures for Control" under the supervision of Prof. Ursula Martin.

References

[1] Samuel Eilenberg and Saunders MacLane. General theory of natural equivalences. *Transactions of the American Mathematical Society*, 58(2):231–294, 1945.

[2] Jean-Yves Girard. Towards a geometry of interaction. In J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, pages 69–108. American Mathematical Society, 1989. Proceedings of the AMS-IMS-SIAM Joint Summer Research Conference, June 14–20, 1987, Boulder, Colorado; Contemporary Mathematics Volume 92.

[3] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–585, 1969.

[4] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Proc. Camb. Phil. Soc.*, 119:447–468, 1995.

[5] Saunders MacLane. *Categories for the Working Mathematician.* Springer, 1971.

[6] Katsuhiko Ogata. *Modern Control Engineering.* Prentice-Hall, 3rd edition, 1997.

[7] Ruth Hardy & Ursula Martin Richard Boulton. A Hoare Logic for Single-Input Single-Output Continuous-Time Control Systems . In *Proceedings 6th International Workshop on Hybrid Systems, Computation and Control*, volume 2623 of *Lecture Notes in Computer Science*, pages 113–125. Springer, 2003.

[8] J. J. M. M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theor. Comput. Sci.*, 308(1-3):1–53, 2003.

Incorporating Formal Methods in the Design Flow of DSP Systems

Behzad Akbarpour and Sofiène Tahar

Dept. of Electrical and Computer Engineering, Concordia University

1455 de Maisonneuve W., Montreal, Quebec, H3G 1M8, Canada

Email: {behzad,tahar}@ece.concordia.ca

Abstract

In this paper we propose a framework for the incorporation of formal methods in the design flow of DSP (Digital Signal Processing) systems in a rigorous way. In the proposed approach we model and verify DSP descriptions at different abstraction levels using higher-order logic based on the HOL (Higher Order Logic) theorem prover. This framework enables the formal verification of DSP designs which in the past could only be done partially using conventional simulation techniques. To this end, we provide a shallow embedding of DSP descriptions in HOL at the floating-point, fixed-point, RTL (Register Transfer Level), and netlist gate levels. We make use of existing formalization of floating-point theory in HOL and introduce a parallel one for fixed-point arithmetic. The high ability of abstraction in HOL allows a seamless hierarchical verification encompassing the whole DSP design path, starting from top level floating- and fixed-point algorithmic descriptions down to RTL, and gate level implementations. We illustrate the new verification framework using different case studies such as digital filters and FFT (Fast Fourier Transform) algorithms.

1 Introduction

Digital system design is characterized by ever increasing system complexity that has to be implemented within reduced time, resulting in minimum costs and short time-to-market. These characteristics call for a seamless design flow that allows to perform the design steps on the highest suitable level of abstraction. For most digital signal processing systems, the design has to result in a fixed-point implementation. This is due to the fact that these systems are sensitive to power consumption, chip size and price per device. Fixed point realizations outperform floating-point realizations by far with regard to these criteria. Figure 1 illustrates a general DSP design flow as used nowadays in leading industry design projects.

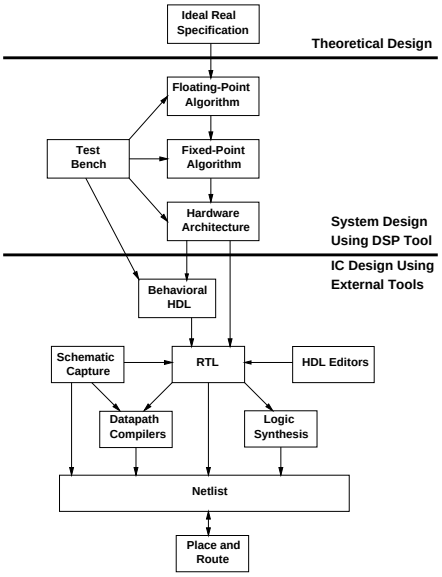


Figure 1: DSP design flow

The design of digital signal processing systems starts from an ideal real number specification. In theoretical analysis of digital systems, we generally assume that signal values and system coefficients are represented

in the real number system and expressed to infinite precision. This allows to ignore the effects of finite wordlengths and fixed exponents and to abstract from all implementation details. When implemented in special-purpose digital hardware or as a computer algorithm, we must represent signals and coefficients in some digital number system that must always be of finite precision. In this case, attention must be paid to the effects of using finite register lengths to represent all relevant design parameters [26]. Despite the advantages offered by digital networks, there is an inherent accuracy problem associated with digital signal processing systems, since the signals are represented by a finite number of bits and the arithmetic operations must be carried out with an accuracy limited by the finite word length of the number representation. Depending on the type of arithmetic used in the system algorithm, the type of quantization used to reduce the word length to a desired size, and the exact system structure used, one can generally estimate how system performance is affected by these finite precision effects. There are several types of arithmetics used in the implementation of digital systems. Among the most common are floating-point and fixed-point. At the floating- and fixed-point levels, all operands are represented by a special format or assigned a fixed word length and a fixed exponent, while the control structure and the operations of the ideal program remain unchanged. The transformation from real (numbers) to floating- and fixed-point is quite tedious and error-prone. On the implementation side, the fixed-point model of the algorithm has to be transformed/synthesized into the best suited target description, either using a hardware description language (HDL) or a programming language. Meeting the above (sometimes conflicting) requirements is a great challenge in any DSP design project.

The above design process can be aided by a number of specialized CAD tools such as SPW (Cadence) [7], CoCentric (Synopsys) [8], Matlab-Simulink (Mathworks) [24], and FRIDGE (Aachen UT) [22]. The conformance of the fixed-point implementation with respect to the descriptions in floating-point or real algorithm on one hand, and the RT (Register Transfer) and gate levels on the other hand is verified by simulation techniques. Simulation, however, is known to provide partial verification as it cannot cover all design errors, especially for large systems. On the other hand, adopting formal verification in system design generally means using methods of mathematical proof rather than simulation to ensure the quality of the design, to improve the robustness of a design and to speed up the development.

In this paper we propose a methodology for applying formal methods to the design flow of DSP systems in a rigorous way. The corresponding commutating diagram is shown in Figure 2.

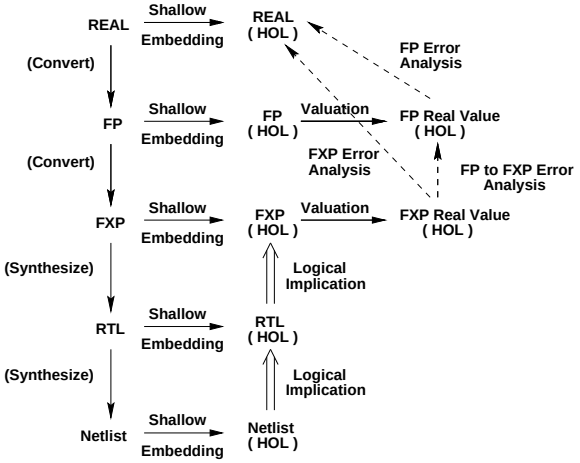


Figure 2: Proposed DSP specification and verification approach

Thereafter, we model the ideal real specification of the DSP algorithms and the corresponding floating-point (FP) and fixed-point (FXP) representations as well as the RT and gate level implementations as predicates in higher-order logic. The overall methodology for the formal specification and verification of DSP algorithms will be based on the idea of shallow embedding of languages [6] using the HOL theorem proving environment[13]. In the proposed approach, we first focus on the transition from real to floating- and fixed-point levels. For this, we make use of existing theories in HOL on the construction of real [14] and complex [17] numbers, the formalization of IEEE-754 standard [18] based floating-point arithmetic [15, 16], and the formalization of fixed-point arithmetic [4]. We use valuation functions to find the real values of the floating- and fixed-point DSP outputs and define the error as the difference between these values and the

corresponding output of the ideal real specification. Then we establish fundamental lemmas on the error analysis of floating- and fixed-point roundings and arithmetic operations against their abstract mathematical counterparts. Finally, based on these lemmas, we derive expressions for the accumulation of roundoff error in floating- and fixed-point DSP algorithms using recursive definitions and initial conditions. While theoretical work on computing the errors due to finite precision effects in the realization of DSP algorithms with floating- and fixed-point arithmetics has been extensively studied since the late sixties [21], this paper contains the first formalization and proof of this analysis using a mechanical theorem prover, here HOL. The formal results are found to be in good agreement with the theoretical ones.

After handling the transition from real to floating- and fixed-point levels, we turn to the HDL representation. At this point, we use well known techniques to model the DSP design at the RTL level within the HOL environment. The last step is to verify this level using a classical hierarchical proof approach in HOL [25]. In this way, we hierarchically prove that the DSP RTL implementation implies the high level fixed-point algorithmic specification, which has already been related to the floating-point description and the ideal real specification through the error analysis. The verification can be extended, following similar manner, down to gate level netlist either in HOL or using other commercial verification tools as depicted in Figure 2.

The rest of the paper is organized as follows: Section 2 describes the fixed-point arithmetic and the details of its formalization in higher-order-logic. Section 3 describes the error analysis of digital filters using HOL theorem proving. Section 4 presents the verification of FFT algorithms in HOL from real specification to gate level implementation. Finally, Section 5 concludes the paper and outlines the future research directions.

2 Formalization of Fixed-Point Arithmetic in HOL

In this section, the objective is to formalize the fixed-point arithmetic in higher-order logic as a basis for checking the correctness of the implementation of DSP designs against higher level algorithmic descriptions in floating-point and fixed-point representations.

2.1 Fixed-Point Numbers

We first describe the fixed-point arithmetic definitions on which we base our formalization. Unlike floating-point arithmetic which is standardized in IEEE-754 [18] and IEEE-854 [19], current fixed-point arithmetic does not follow any particular standard and depends on the tool and the language used to design the DSP chip. While we tried to keep these definitions as general as possible, the fixed-point numbers format, arithmetic operations, overflow and quantization modes, and exception handling adopted are to some extent influenced by the fixed-point arithmetic defined by Cadence SPW [7] and Synopsys SystemC [27].

A fixed-point number has a fixed number of binary digits and a fixed position for the decimal point with respect to that sequence of digits. Fixed-point numbers can be either unsigned (always positive) or signed (in two's complement representation). Fixed-point numbers are expressed as a pair consisting of a binary string and a set of attributes. The attributes specify how the binary string is interpreted. Generally, the attributes consists of three parameters to represent the total word length, the integer word length, and the sign format. Operations performed on fixed-point data types are done using arbitrary and full precision. After the operation is complete, the resulting operand is cast to fit the fixed-point data type object. The casting operation applies the quantization behavior of the target object to the new value and assigns the new value to the target object. Then, the appropriate overflow behavior is applied to the result of the process which gives the final value. In addition to the parameters corresponding to the input operands and output result, the arithmetic operations take specific parameters defining the overflow and quantization modes. These parameters are the Quantization mode, Overflow mode, and Number of saturated bits. Quantization effects are used to determine what happens to the LSBs of a fixed-point type when more bits of precision are required than are available. The quantization modes are: Quantization to Plus Infinity, Quantization to Zero, Quantization to Minus Infinity, Quantization to Infinity, Convergent Quantization, Truncation, and Truncation to Zero. In addition to quantization modes, we can use overflow modes to approximate a higher range for fixed-point operations. Usually, overflow occurs when the result of an operation is too large or too small for the available bit range. Specific overflow modes can then be implemented to reduce the loss of data. Overflow modes are: Saturation, Saturation to Zero, Symmetrical Saturation, Wrap-Around, and Sign Magnitude Wrap-Around.

2.2 Fixed-point Formalization in HOL

In this section, we present formalization of the fixed-point arithmetic in higher-order logic, based on the general purpose HOL theorem prover. HOL's basic types include the natural numbers and booleans. It also includes other specific extensions like John Harrison's reals library [14] which proved to be essential for our fixed-point arithmetic formalization.

Fixed point numbers are modeled in HOL as a pair of elements composed of a bit string and a set of attributes. The bit string is represented by a boolean word and the set of attributes is itself a combination of six elements representing the word length, integer word length, sign type, rounding mode, overflow mode, and the number of saturation bits. The fixed-point numbers are then partitioned using special predicates into signed and un-signed numbers. The validity of a fixed-point number and a set of attributes is defined using special predicates. Then we defined the actual HOL type for the fixed-point numbers. The valuation function is then defined to specify a real value to fixed-point numbers using separate formulas for signed and unsigned numbers. The constants for the smallest and largest fixed-point numbers for a given format together with their corresponding real values are also defined using specific functions. Then, we defined enumerated data types for seven rounding modes and five overflow modes in fixed-point arithmetic. The rounding function is then defined case by case on the rounding and overflow modes. Then, we defined the operations on fixed-point numbers which are performed using the arbitrary precision in real domain and then the result is casted to the output format.

2.3 Verification of Fixed-Point Operations

According to the discussion in the previous section, each fixed-point number has a corresponding real number value. The correctness of a fixed-point operation can be specified by comparing its output with the true mathematical result, using the valuation function *value* that converts a fixed-point to an infinitely precise number. For example, the correctness of a fixed-point adder *fxpAdd* is specified by comparing it with its ideal counterpart +. That is, for each pair of fixed-point numbers *(a,b)*, we compare *value (a)* + *value (b)* and *value (fxpAdd (a,b))*. In other words, we check if the diagram in Figure 3 commutes.

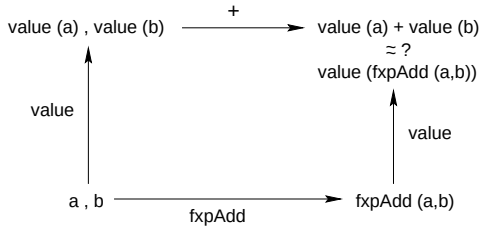


Figure 3: Correctness criteria for fixed-point addition

Therefore, the correctness of fixed-point operations can be specified by comparing the operation's output with the true mathematical result. Since the operations are defined as if they first performed using infinite precision and then the result is rounded to fit in the destination format, the verification of operations is closely related to bounding the error in rounding function. The steps in analysis of fixed-point rounding error in HOL are as follows:

- **Lemmas for Analyzing the Fixed-Point Rounding Operation:** We first proved lemmas concerning with the approximation of a real number with a fixed-point number. We proved that in a finite nonempty set of fixed-point numbers we can find the best approximation to a real number based on a given valuation function. Then, we proved that the chosen best approximation to a real number satisfying a property p from a finite and non empty set of fixed-point numbers is unique and is itself a member of the set. Finally, we proved that the chosen best approximation to a real number satisfying a property p from the set of all valid fixed-point numbers with a given attributes is itself a valid fixed-point number.

- **Rounding Error in Fixed-Point Arithmetic Operations:** Then, we defined the error resulting from rounding a real number to a fixed-point value. Then, we established the first main theorems on

the correctness of fixed-point arithmetic operations. According to these theorems, if the input fixed-point operands and the output attributes are valid then the result of fixed-point operations is valid. Also the result of the operations is related to the real result considering the error.

- **General Fixed-Point Error Bound Theorem:** In the next step, we established the second main theorem on fixed-point rounding error analysis which concerns bounding the error. According to this theorem, the error in rounding a real number which is in the range representable by a given set of attributes X is less than the quantity $1/2^{fracbits(X)}$, where *fracbits* is the number of bits that are to the right of the binary point in the given fixed-point format X. To explain the theorem, we consider the distribution of fixed-point number on the real axis as shown by Figure 4. Thereafter, the representable range of fixed-point numbers is divided into 2^N equispaced quantization steps with the distance between two successive steps equal to $1/2^b$. Suppose that $x \in \mathbb{R}$ is approximated by a fixed-point number a. The position of these values are labeled in the figure. The error $\mid x - a \mid$ is hence less than the length of one interval, or $1/2^b$, as mentioned in the second theorem. The details can be found in [4].

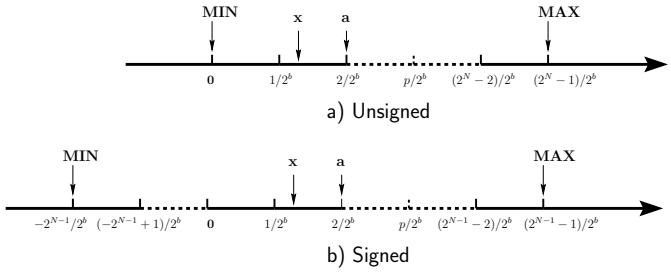


Figure 4: Fixed-Point Values on the Real Axis

3 Error Analysis of Digital Filters

Digital filters are a particularly important class of DSP (Digital Signal Processing) systems. A digital filter is a discrete time system that transforms a sequence of input numbers into another sequence of output, by means of a computational algorithm [20]. Digital filters are used in a wide variety of signal processing applications, such as spectrum analysis, digital image and speech processing, and pattern recognition. Due to their well-known advantages, digital filters are often replacing classical analog filters. The three distinct and most outstanding advantages of the digital filters are their flexibility, reliability, and modularity. Excellent methods have been developed to design these filters with desired characteristics. The design of a filter is the process of determination of a transfer function from a set of specifications given either in the frequency domain, or in the time domain, or for some applications, in both. When a digital filter is realized with floating-point or fixed-point arithmetics, errors and constraints due to finite word length are unavoidable. In this section, as the first case study for our verification methodology, we show how these errors can be mechanically analyzed using the HOL theorem prover. We first model the ideal real filter specification and the corresponding floating-point and fixed-point implementations as predicates in higher-order logic. We use valuation functions to find the real values of the floating-point and fixed-point filter outputs and define the error as the difference between these values and the corresponding output of the ideal real specification. Fundamental analysis lemmas have been established to derive expressions for the accumulation of roundoff error in parametric Lth-order digital filters, for each of the three basic forms of realization: direct, parallel, and cascade. The HOL formalization and proofs are found to be in a good agreement with existing theoretical paper-and-pencil counterparts.

3.1 Error Analysis Models

In this section we introduce the fundamental error analysis theorems [28, 11], and the corresponding lemmas in HOL for the floating-point [15, 16] and fixed-point arithmetics [4]. These theorems are then used in the next sections as a model for the analysis of the roundoff error in digital filters.

In analyzing the effect of floating-point roundoff, the effect of rounding will be represented multiplicatively. Letting $*$ denote any of the arithmetic operations +, -, $\times$, /, it is known [11, 28] that, if p represents the

precision of the floating-point format, then

$$fl\ (x\ *\ y)\ =\ (x\ *\ y)(1\ +\ \delta),\ \ \text{where}\ \ |\delta|\ \leq\ 2^{-p}. \tag{1}$$

The notation fl (.) is used to denote that the operation is performed using floating-point arithmetic. The theorem relates the floating-point arithmetic operations such as addition, subtraction, multiplication, and division to their abstract mathematical counterparts according to the corresponding errors.

While the rounding error for floating-point arithmetic enters into the system multiplicatively, it is an additive component for fixed-point arithmetic. In this case the fundamental error analysis theorem for fixed-point arithmetic operations against their abstract mathematical counterparts can be stated as

$$fxp\ (x\ *\ y)\ =\ (x\ *\ y)\ +\ \epsilon,\ \ \text{where}\ \ |\epsilon|\ \leq\ 2^{-fracbits\ (X)}. \tag{2}$$

The notation fxp (.) is used to denote that the operation is performed using fixed-point arithmetic. We have proved equations (1) and (2) as theorems in higher-order logic within HOL. The theorems are proved under the assumption that there is no overflow or underflow in the operation result. This means that the input values are scaled so that the real value of the result is located in the ranges defined by the maximum and minimum representable values of the given floating-point and fixed-point formats.

3.2 Error Analysis of Digital Filters in HOL

In this section, the principal results for the roundoff accumulation in digital filters using the mechanized theorem proving are derived and summarized. We shall employ the models for the floating-point and fixed-point roundoff errors in HOL presented in the previous section.

The class of digital filters considered in this paper is that of linear constant coefficient filters specified by the difference equation:

$$w_n\ =\ \sum_{i=0}^{M} b_i\ x_{n-i}\ -\ \sum_{i=1}^{L} a_i\ w_{n-i} \tag{3}$$

where $\{x_n\}$ is the input sequence and $\{w_n\}$ is the output sequence. L is the order of the filter, and M can be any positive number less than L. There are three basic forms of realizing a digital filter, namely the direct, parallel, and cascade forms. If the output sequence is calculated by using the equation (3), the digital filter is said to be realized in the direct form. In parallel form, the entire filter is visualized as the parallel connection of the simpler filters of a lower order. In cascade form, the filter is visualized as a cascade of lower filters.

There are three common sources of errors associated with the filter of the equation (3), namely [23]: input quantization, coefficient inaccuracy, and round-off accumulation. Therefore, for the digital filter of the equation (3) the actual computed output reference is in general different from $\{w_n\}$. We denote the actual floating-point and fixed-point outputs by $\{y_n\}$ and $\{v_n\}$, respectively. Then, we define the corresponding errors at the nth output sample as e_n, e'_n and e''_n, where e_n and e'_n are defined as the errors between the actual floating-point and fixed-point implementations and the ideal real specification, respectively. e''_n is the error in the transition from the floating-point to fixed-point levels.

The corresponding flowgraph showing the effect of roundoff error using the fundamental error analysis theorems according to the equations (1) and (2), is given by Figure 5, which also indicates the order of the calculation. The quantities $\delta, \epsilon_{n,k}, \zeta_{n,k}, \eta_{n,k}$, and ξ_n in Figure 5 are errors caused by the floating-point roundoff at each arithmetic step. The corresponding error quantities for the fixed-point roundoff (shown in parentheses) are $\delta'_{n,k}, \epsilon'_{n,k}, \zeta'_{n,k}, \eta'_{n,k}$, and ξ'_n.

For the error analysis, we need to calculate the y_n and v_n sequences, and compare them with the ideal output sequence w_n specified by the equation (3) to obtain the corresponding errors e_n, e'_n, and e''_n. Therefore, we derived the difference equations for the errors between the different levels showing the accumulation of the roundoff error in HOL. Similar analysis is performed for the parallel and cascade forms of realization based on the corresponding error flowgraphs. The details can be found in [3].

4 Verification of FFT algorithm

The fast Fourier transform (FFT) [5, 9] is an algorithm to compute the discrete Fourier transform with a substantial time saving over conventional methods. FFT algorithms are based on the fundamental principle

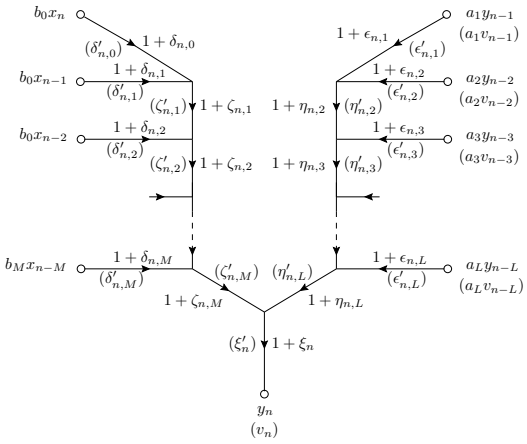


Figure 5: Error flowgraph for Lth-order filter (Direct form)

of decomposing the computation of the discrete Fourier transform of a sequence of length N into successively smaller discrete Fourier transforms. The manner in which this principle is implemented leads to a variety of different algorithms, all with comparable improvements in computational speed. Two basic classes of FFT algorithms are the decimation-in-time (DIT) [10] and decimation-in-frequency (DIF) [12]. As the second case study in this project, we considered the formal verification of FFT algorithms. We used our methodology to derive expressions for the accumulation of roundoff error in floating- and fixed-point FFT algorithms by recursive definitions and initial conditions, considering the effects of input quantization and inaccuracy in the coefficients. Based on the extensively studied theoretical work on computing the errors due to finite precision effects in the realization of FFT algorithms with floating- and fixed-point arithmetics, we performed a similar analysis using the HOL theorem proving environment. The formal results are found to be in good agreement with the theoretical ones.

4.1 Error Analysis of FFT Algorithms in HOL

In this section, the principal results for roundoff accumulation in FFT algorithms using HOL theorem proving are derived and summarized. For the most part, the following discussion is phrased in terms of the decimation-in-frequency form of radix-2 algorithm. The results, however, are applicable with only minor modification to the decimation-in-time form. Furthermore, most of the ideas employed in the error analysis of the radix-2 algorithms can be utilized in the analysis of other algorithms.

Let $\{A_k(p)\}_{p=0}^{N-1}$ denote the N complex numbers calculated at the kth step. Then the decimation in frequency (DIF) FFT algorithm can be expressed as [21]

$$A_{k+1}(p) = \begin{cases} A_k(p) + A_k(p + 2^{m-1-k}) & \text{if } p_k = 0 \\ [A_k(p - 2^{m-1-k}) - A_k(p)]\ w_k(p) & \text{if } p_k = 1 \end{cases} \tag{4}$$

where $w_k(p)$ is a power of the principle roots of unity, and $N = 2^m$, where m is an integer value. Equation (4) is carried out for $k = 0, 1, 2, \ldots, m-1$, with $A_0(p) = x(p)$, where $\{x(n)\}_{n=0}^{N-1}$ is the set of input sequence. It can be shown [12] that at the last step $\{A_m(p)\}_{p=0}^{N-1}$ are the discrete Fourier coefficients in rearranged order.

There are three common sources of errors associated with the FFT algorithms, namely: input quantization, coefficient inaccuracy, and round-off accumulation. Therefore, the actual array computed by using equation (4) is in general different from $\{A_k(p)\}_{p=0}^{N-1}$. We denote the actual floating- and fixed-point computed arrays by $\{A'_k(p)\}_{p=0}^{N-1}$ and $\{A''_k(p)\}_{p=0}^{N-1}$, respectively. Then, we define the corresponding errors of the pth element at step k as $e_k(p), e'_k(p)$, and $e''_k(p)$, where $e_k(p)$ and $e'_k(p)$ are defined as the error between the actual floating- and fixed-point implementations and the ideal real specification, respectively. $e''_k(p)$ is the error in transition from floating- to fixed-point levels.

In equation (4) the $\{A_k(p)\}$ are complex numbers, so their real and imaginary parts are calculated separately. We define the real and imaginary of $A_k(p)$ and $w_k(p)$ as $B_k(p), C_k(p)$ and $U_k(p), V_k(p)$, respectively. Similarly, we express the real and imaginary parts of $A'_k(p)$, $B'(p)$ and $C'_k(p)$, and $A''_k(p)$, $B''_k(p)$ and $C''_k(p)$,

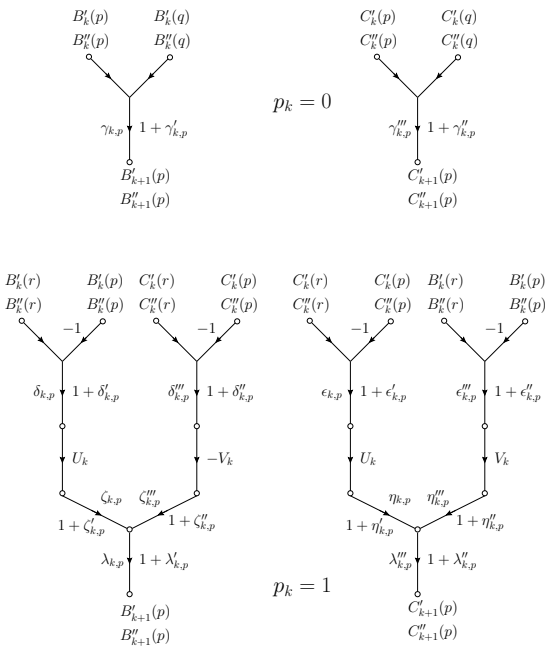


Figure 6: Error flowgraph for decimation-in-frequency FFT

using the floating- and fixed-point operations, respectively. The corresponding error flowgraph showing the effect of roundoff error using the fundamental floating- and fixed-point error analysis theorems according to the equations (1) and (2), respectively, is given in Figure 6, which also indicates the order of the calculation.

The quantities γ', γ'', δ', δ'', ϵ', ϵ'', ζ', ζ'', η', η'', λ', and λ'' in Figure 6 are errors caused by floating-point roundoff at each arithmetic step. The corresponding error quantities for fixed-point roundoff are γ, γ''', δ, δ''', ϵ, ϵ''', ζ, ζ''', η, η''', λ, and λ'''.

Therefore, we derived in HOL, expressions for the accumulation of roundoff error for FFT algorithm by recursive equations and initial conditions. The details can be found in [2].

4.2 FFT Design Implementation Verification

In this section, we describe the application of the proposed approach for the verification in HOL of the transition from real, floating- and fixed-point specifications to RTL implementation of an FFT algorithm. We have chosen the case study of a radix-4 pipelined 64-point complex FFT core available as VHDL RTL model in the Xilinx Coregen library [29]. All proofs have been conducted in HOL, hence establishing a correctness of the FFT design implementation with respect to its high level algorithmic specifications. Figure 7 shows the overall block diagram of the Radix-4 64-point pipelined FFT design.

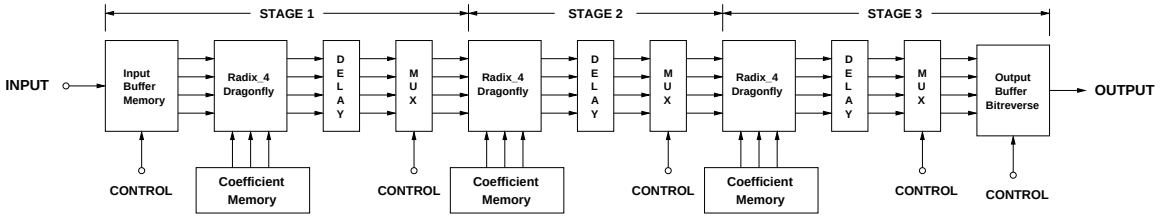


Figure 7: Radix-4 64-point pipelined FFT implementation

The basic elements are memories, delays, multiplexers, and dragonflies. In general, the 64-point pipelined FFT requires the calculation of three radix-4 dragonfly ranks. Each radix-4 dragonfly is a successive combination of a radix-4 butterfly with four twiddle factor multipliers. The FFT core accepts naturally ordered data on the input buses in a continuous stream, performs a complex FFT, and streams out the DFT samples on the output buses in a natural order. These buses are respectively the real and imaginary components of

the input and output sequences. An internal input data memory controller orders the data into blocks to be presented to the FFT processor. The twiddle factors are stored in coefficient memories. The real and imaginary components of complex input and output samples and the phase factors are represented as 16-bit 2's complement numbers. The unscrambling operation is performed using the output bit-reversing buffer.

In HOL, we first modeled the RTL description of a radix-4 butterfly as a predicate in higher-order logic. The block takes a vector of four complex input data and performs the operations, to generate a vector of four complex output signals. The real and imaginary parts of the input and output signals are represented as 16-bit Boolean words. We defined separate functions in HOL for arithmetic operations such as addition, subtraction, and multiplication on complex two's complement 16-bit Boolean words. Then, we built the complete butterfly structure using a proper combination of these primitive operations.

Thereafter, we described a radix-4 dragonfly block as a conjunction of a radix-4 butterfly and four 16-bit twiddle factor complex multipliers. Finally, we modeled the complete RTL description of the radix-4 64-point structure in HOL. The FFT block is defined as a conjunction of 48 instantiations of radix-4 dragonfly blocks. Proper time instances of the input and output signals are applied to each block, according to Figure **??**.

Following similar steps, we described the radix-4 64-point FFT structures as fixed-point, floating-point, and real domains in HOL using the corresponding complex data types and arithmetic operations.

The formal verification of the radix-4 decimation in frequency FFT algorithm case study was performed based on the commutating diagram in Figure 2, in that we proved hierarchically that the FFT Netlist implies the FFT RTL; and then proved that the FFT RTL description implies the corresponding fixed-point model. The proof of the FFT block is then broken down into the corresponding proof of the dragonfly block, which itself is broken down into the proofs of butterfly and primitive arithmetic operations. We used the data abstraction functions to convert a complex vector of 16-bit two's complement Boolean words into the corresponding fixed-point vector.

Then, we proved three theorems encompassing the error analysis of the radix-4 decimation in frequency FFT algorithm. The first lemma represents the error between the real number specification and the floating-point specification. The second lemma represents the error between the real number and the fixed-point specifications. The third lemma represents the error between floating-point and fixed-point specifications. According to these lemmas, the floating-point and fixed-point implementations and the real specification of a radix-4 decimation in frequency FFT algorithm are related to each other based on the corresponding data abstraction, and error analysis functions.

Finally, using the obtained theorems, we easily deduced our ultimate theorem proving the correctness of the real specification from the RTL implementation, taking into account the error analysis computed beforehand. A complete list of the derived HOL definitions and theorems can be found in [1].

5 Conclusions

In this paper, we described a methodology for the formal specification and verification of DSP systems designs at different abstraction levels. We proposed a shallow embedding of DSP descriptions at different levels in HOL. For the verification of the transition from floating- to fixed-point levels, we proposed an error analysis approach in which we consider the effects of finite precision in the implementation of DSP systems. These include errors due to the quantization of input samples and system coefficients, and also roundoff accumulation in arithmetic operations. The verification from fixed-point to RTL and netlist levels is performed using traditional hierarchical verification in HOL. In this paper we demonstrated our methodology using the case studies of digital filters and the fast Fourier transform algorithms. The approach covers three basic forms (direct, parallel, and cascade) of realization of the digital filters, and the two canonical forms (decimation-in-time, and decimation-in-frequency) of realization of the FFT algorithm using real, floating-, and fixed-point arithmetic as well as their RT implementations, each entirely specified in HOL. We proved lemmas to derive expressions for the accumulation of roundoff error in floating- and fixed-point designs compared to the ideal real specification. Then we proved that the FFT RTL implementation implies the corresponding specification at the fixed-point level using classical hierarchical verification in HOL, hence bridging the gap between hardware implementation and high levels of mathematical specification. In this work we also have contributed to the upgrade and application of established real, complex real, floating- and fixed-point theories in HOL to the analysis of errors due to finite precision effects, and applied them on the realization of the FFT algorithms. Error analyses using theoretical paper-and-pencil proofs did exist since the late sixties while design verification is exclusively done by simulation techniques. We believe this is the

first time a complete formal framework has been proposed for the specification and verification of the DSP algorithms at different levels of abstraction. The methodology presented in this paper opens new avenues in using formal methods for the verification of digital signal processing (DSP) systems as complement to traditional theoretical (analytical) and simulation techniques. We are currently investigating the verification of complex wired and wireless communication systems, whose building blocks, heavily make use of several instances of the FFT algorithms. As a future work, we also plan to extend the error analyses to cover worst-case, average, and variance errors. Finally, we plan to link HOL with computer algebra systems to create a sound, reliable, and powerful system for the verification of DSP systems.

References

[1] B. Akbarpour, "Modeling and Verification of DSP Designs in HOL," Ph.D. Thesis, Concordia University, Department of Electrical and Computer Engineering, Montreal, Canada, March 2005.

[2] B. Akbarpour and S. Tahar, "A Methodology for the Formal Verification of FFT Algorithms in HOL," In Formal Methods in Computer-Aided Design, LNCS 3312, pp. 37-51, Springer-Verlag, 2004.

[3] B. Akbarpour and S. Tahar, "Error Analysis of Digital Filters using Theorem Proving," In Theorem Proving in Higher Order Logics, LNCS 3223, pp. 1-16, Springer-Verlag, 2004.

[4] B. Akbarpour, S. Tahar, and A. Dekdouk, "Formalization of Fixed-Point Arithmetic in HOL," Formal Methods in Systems Design, 27: 173-200, 2005.

[5] E. O. Brigham, "The Fast Fourier Transform," Prentice Hall, 1974.

[6] R. Boulton, A. Gordon, M. Gordon, J. Harrison, J. Herbert, and J. Van-Tassel, "Experience with Embedding Hardware Description Languages in HOL," In Theorem Provers in Circuit Design, pp. 129-156, North-Holland, 1992.

[7] Cadence Design Systems, Inc., "Signal Processing WorkSystem (SPW) User's Guide," USA, July 1999.

[8] Synopsys, Inc., "CoCentric^{TM} System Studio User's Guide," USA, Aug. 2001.

[9] W. T. Cochran et. al., "What is the Fast Fourier Transform," IEEE Transactions on Audio and Electroacoustics, AU-15: 45-55, Jun. 1967.

[10] J. W. Cooley, J. W. Tukey, "An Algorithm for Machine Calculation of Complex Fourier Series," Mathematics of Computation, 19: 297-301, Apr. 1965.

[11] G. Forsythe and C. B. Moler, "Computer Solution of Linear Algebraic Systems," Prentice-Hall, 1967.

[12] W. M. Gentleman and G. Sande, "Fast Fourier Transforms - For Fun and Profit," In AFIPS Fall Joint Computer Conference, Vol. 29, pp. 563-578, Spartan Books, Washington, DC, 1966.

[13] M. J. C. Gordon and T. F. Melham, "Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic," Cambridge University Press, 1993.

[14] J. R. Harrison, "Constructing the Real Numbers in HOL," Formal Methods in System Design, 5 (1/2): 35-59, 1994.

[15] J. R. Harrison, "A Machine-Checked Theory of Floating-Point Arithmetic," In Theorem Proving in Higher Order Logics, LNCS 1690, pp. 113-130, Springer-Verlag, 1999.

[16] J. R. Harrison, "Floating-Point Verification in HOL Light: The Exponential Function," Formal Methods in System Design, 16 (3): 271-305, 2000.

[17] J. R. Harrison, "Complex Quantifier Elimination in HOL," In Supplemental Proceedings of the International Conference on Theorem Proving in Higher Order Logics, pp. 159-174, Edinburgh, Scotland, UK, Sep. 2001.

[18] IEEE, Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985, The Institute of Electrical and Electronic Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA, 1985.

[19] The Institute of Electrical and Electronic Engineers, Inc., "IEEE, Standard for Radix-Independent Floating-Point Arithmetic," ANSI/IEEE Std 854, USA, 1987.

[20] J. F. Kaiser, "Digital Filters," In System Analysis by Digital Computer, F. F. Kuo and J. F. Kaiser, Eds., pp. 218-285, Wiley, 1966.

[21] T. Kaneko and B. Liu, "Accumulation of Round-Off Error in Fast Fourier Transforms," Journal of Association for Computing Machinery, 17 (4): 637-654, Oct. 1970.

[22] H. Keding, M. Willems, M. Coors, and H. Meyr, "FRIDGE: A Fixed-Point Design and Simulation Environment," In Proceedings Design Automation and Test in Europe Conference, pp. 429-435, Paris, France, February 1998.

[23] B. Liu, and T. Kaneko, "Error Analysis of Digital Filters Realized with Floating-Point Arithmetic," Proceedings of the IEEE, 57: 1735-1747, October 1969.

[24] Mathworks, Inc., "Simulink Reference Manual," USA, 1996.

[25] T. Melham, "Higher Order Logic and Hardware Verification," Cambridge Tracts in Theoretical Computer Science 31, Cambridge University Press, 1993.

[26] A. V. Oppenheim and C. J. Weinstein, "Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform," Proceedings of the IEEE, 60 (8): 957-976, August 1972.

[27] Open SystemC Initiative, "SystemC Language Reference Manual," USA, 2004.

[28] J. H. Wilkinson, "Rounding Errors in Algebraic Processes," Prentice-Hall, 1963.

[29] Xilinx, Inc., "High-Performance 64-Point Complex FFT/IFFT V2.0, Product Specification," USA, Aug. 2000, http://xilinx.com/ipcenter.

On the Formal Analysis of Analog Systems using Interval Abstraction

Mohamed H. Zaki, Ali Habibi, Sofiène Tahar and Guy Bois §

Dept. of Electrical & Computer Engineering, Concordia University
1455 de Maisonneuve W., Montreal, Quebec, H3G 1M8, Canada
§Genie Informatique, Ecole Polytechnique de Montreal
Pavillon Decelles, 5255 Avenue Decelles, Montreal, QC H3T 2B1, Canada
{mzaki,habibi,tahar}@ece.concordia.ca, guy.bois@polymtl.ca

Abstract. Formal methods have been advocated for the verification of digital design where correctness is proved mathematically. In contrast to digital designs, the verification of analog and mixed signal (AMS) systems is a challenging task that requires lots of expertise and deep understanding of their behavior. In this paper, we intend to present an ongoing project for developing an approach for the analysis of analog systems using formal methods. The proposed method is based on the computation of finite-state conservative abstraction of the design. For this purpose, we use abstract interpretation, which is a theoretical framework for the analysis of infinite state systems and in this paper we choose interval arithmetics as the basis of the abstraction. To test and validate our methodology, we have used interval arithmetic tool AWA to describe the behavior of tunnel diode oscillator.

1 Introduction

Formal methods have been advocated for the verification of digital design where correctness is proved mathematically. In contrast to digital designs, the verification of analog and mixed signal (AMS) systems is a challenging task that requires lots of expertise and deep understanding of their behavior. The functionality of analog circuits is defined directly in terms of continuous electrical quantities and is usually sensitive to environment factors like signal noise, current leakage, temperature, etc, in addition to higher order physical effects when designing in deep submicron. Traditionally simulation techniques have been used as the main verification tool in the AMS design process, however the limitation of simulation in terms of coverage affect the confidence in the verification results. Symbolic analysis have been developed as a complementary to numerical simulation, where the effect of parameters variations on the system behavior is analyzed. Although successful, challenging problems (like non linear effects) make this techniques only suitable for simple designs.

The last decade saw the emergence of a new engineering field known as hybrid system theory where researchers have developed formal techniques for the design and analysis automation of systems with real time and continuous

behavior and which are described by a composition of continuous time systems and discrete time systems. Model checking tools such as HyTech [3], CheckMate [2] and d/dt [1] were successful in the verification of hybrid systems with different degrees of complexity. Motivated by the success of the application of formal verification methods for real-time and hybrid systems, researchers started in recent years studying the applicability of such techniques for the verification of analog and mixed signal systems. In this paper, we intend to present an ongoing project for developing an approach for the analysis of analog systems using formal methods. The proposed method is based on the computation of finite-state conservative abstraction of the design. The methodology should guarantee that the verification on the abstract model is conservative. For this purpose, we use abstract interpretation, which is a theoretical framework for the analysis of infinite state systems and in this paper we choose interval arithmetics as the basis of the abstraction. In the hybrid system theory context, abstract interpretation was only used with simple systems with linear dynamics while other heuristic methods were applied for the abstraction of the nonlinear dynamics. In contrast, we propose a methodology for abstracting nonlinear dynamics using abstract interpretation, which will guarantee property preservation. To test and validate our methodology, we have used interval arithmetic tool AWA [10] to describe the behavior of tunnel diode oscillator.

The rest of the paper is organized as follows: In section 2, we give a brief overview of the project, followed by some preliminaries definitions in section 3. The abstraction methodology is described in section 4, with primary experimental results in 5. Related work is presented in 6 and finally, we conclude the paper with section 7.

2 Methodology

We present in this section our methodology for modeling and verification of continuous time systems. The basic idea is to adopt a multi abstraction technique in order to generate a compact model that can be automatically verified using model checking techniques, yet a conservative one as we need to guarantee property preservation. We start by solving the system of differential equations symbolically. As a start, we choose interval based methods proposed by Moore [11] which was developed to guarantees solution inclusion for differential equations, and used by Cousot and Cousot in software execution analysis [7]. The symbolic execution is monitored by temporal properties with predicates representing properties on the continuous state variables. These predicates form the base of the abstract model which is generated at the back end. Each abstract state is represented by region representing a conjunction of satisfied predicates. The advantage of this method is the accuracy of the analysis as symbolic methods are more accurate than qualitative methods avoiding the imprecision of these techniques. On the other hand, generation of the abstract model by monitoring the solution, once proved sound and conservative, avoids the cost of expensive

model checking of the whole discretized continuous behavior. Figure 1 gives an overview about the methodology.

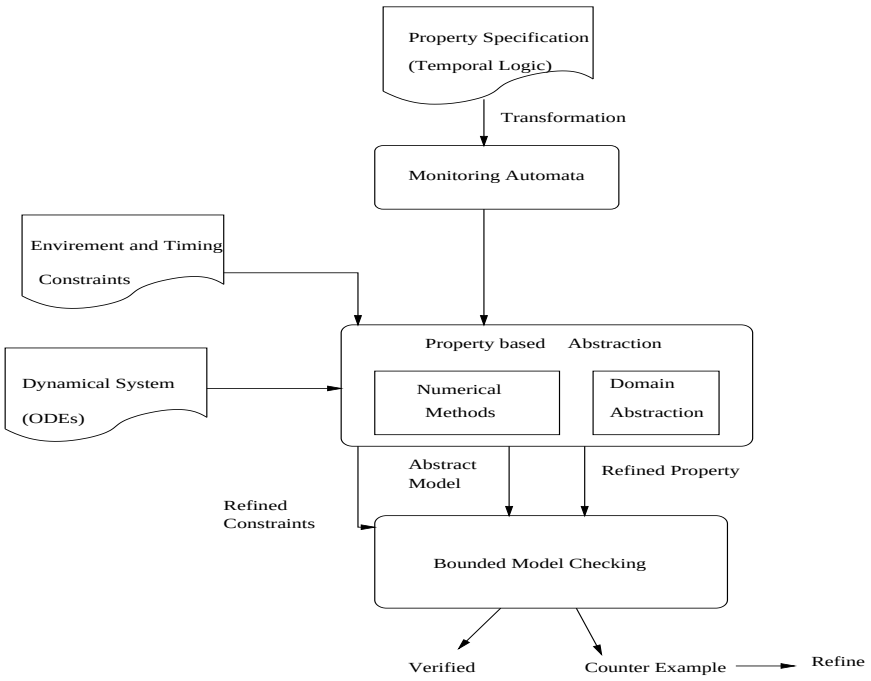


Fig. 1. Proposed verification methodology for continuous systems

In this paper, we focus only on part concerning the solution of differential equation using interval arithmetic. We demonstrate how to use this theory in order to infer reachability properties of non linear dynamical systems.

3 Modeling Continuous Systems

In this section we describe the class of non linear continuous systems represented by a system of ordinary differential equations. This modeling is suitable for describing analog systems behavior.

Definition 1. *A continuous dynamical system is a triple* $CS = (\mathbb{X}, \mathbb{X}_0, f)$ *with* $\mathbb{X} \subseteq \boldsymbol{R}^n$ *is the continuous state space,* $\mathbb{X}_0 \subseteq \mathbb{X}$ *and* $f : \mathbb{X} \to \mathbb{R}^n$ *is the continuous vector field.*

The behavior of a continuous dynamical system is governed by the differential equation $\dot{x} = f(x)$. We assume that the differential equation has a unique solution for each initial value $X(0) \in \mathbb{X}_0$, where $\mathbb{X}_0 \subseteq \mathbb{X}$ is the set of initial

continuous space. The continuous system is deterministic in the sense that from a given point it generates a unique trajectory.

Definition 2. *The semantics of a continuous dynamical system* $CS = (\mathbb{X}, \mathbb{X}_0, f)$ *over a continuous time period (an interval)* $T_c = [\tau_0, \tau_1] \subseteq \mathbb{R}^+$ *can be described as Trajectory. A trajectory of a continuous dynamical system is a continuous behavior* $\Phi_x : T_c \to \mathbb{X}$ *for* $x \in \mathbb{X}_0$ *such that* $\Phi_x(t)$ *is the solution of equation* $\dot{x} = f(x)$ *with initial condition* $\Phi_x(0) = x$ *and* $t \in T_C$*, is a time point and* T_c *is the continuous time domain.*

We can present the behavior of the continuous system by a transition system as follows:

Definition 3. *A dynamical continuous system can be considered as a transition system* $T_c = (Q, Q_0, \sigma, L)$ *where:*

- $q \in Q$ *is a configuration* (x, Γ)*, where* $x \in \mathbb{X}$ *and set of intervals* $\Gamma = \{t_0, \ldots\}$ *such that* $\cup_{i \geq 0} t_i \subseteq \mathbb{R}^+$*, where* $i \subseteq \mathbb{N} \cup \{\infty\}$*. We have* $t_1, t_2 \in \Gamma$ *for* $\Phi_{x'}(t_1) = \Phi_{x''}(t_2) = x$ *and* $x', x'' \in \mathbb{X}_0$ *.*
- $q \in Q_0$*, when* $t_0 \in \Gamma$ *and* $\Phi_x(t_0) = x$ *and* t_0 *is the interval* $[0, 0]$*,*
- L *is an interpretation function such that* $L : Q \to \mathbb{R}^n \times 2^{\mathbb{R}^+}$*.*
- $\sigma \subseteq Q \times Q$ *is a transition relation such that* $(q_n, q_m) \in \sigma$ *iff* $\exists t_n \in \Gamma_n$*,* $\exists t_m \in \Gamma_m$*.* $t_n < t_m$ *and* $\lim_{t_n \to t_m} \Phi_x^{q_n}(t_n) = \Phi_x^{q_m}(t_m)$*,* $x \in \mathbb{X}_0$*.*

The set of reachable states of a continuous system can be defined as follows:

Definition 4. *For a continuous dynamical system, the set of reachable states* $Reach \in \Gamma$ *can be defined as:*

- $Reach^{(0)} := Q_0$
- $Reach := \{q' \in Q | \exists q \in Reach^{(0)}, t \in L_\Gamma(q'), x' = L_x(q'), x = L_x(q)$ *such that* $\Phi_x(t) = x'\}$

Given a transition system C and a set of (un)safe states $B \subseteq \mathbb{X}$, whether $Reach \cap B$ is empty or not.

Remark 1. We have considered Γ associated with each state, as a set of closed time intervals and with a slight abuse of notations, we have referred to a trajectory between two time points as

$$\Phi_x : I_i \to \mathbf{R}^n$$

$$: [t_1, t_2] \mapsto x$$

We can described a discrete system as a transition system as follows:

Definition 5. *A transition system is a triple* $DS = (S, S_0, \sigma)$ *with* S *is the set of states,* $S_0 \subseteq S$ *is the set of initial states and* $\sigma \subseteq S \times S$ *is a relation capturing transitions.*

Definition 6. *The semantics of discrete system can be defined as a set of traces, where trace of a transition system* $DS = (S, S_0, \sigma)$ *is a sequence* $\pi : \boldsymbol{N} \rightarrow S$ *such that* $\pi(0) \in S_o$ *and* $\forall k \geq 0 : (\pi(k), \pi(k+1)) \in \pi$ *and* $\boldsymbol{N}$ *is the natural number domain.*

Usually the analytic solution for the differential equation is not possible and approximate methods are used instead. Given a CS with a configuration $\mathbf{C}$; to describe a correspondence between discrete evolution $\pi : \mathbb{N} \rightarrow \mathbf{C}$ and continuous evolution $\Phi : [0, \infty) \rightarrow \mathbf{Q}$, Tiwari *et.al* [31] defined the notion of sufficient complete discretization. This can be understood as sampling criteria that captures all the different states in the given continuous evolution. It can be formally defined as follows:

Definition 7. *A discrete evolution* $\pi : \mathbb{N} \rightarrow \boldsymbol{C}$ *is a sufficiently complete discretization of a continuous evolution* $\Phi : [0, \infty) \rightarrow \boldsymbol{Q}$, *if* $\pi(i) = \Phi^q_{x_0}(t_i) = x$ *where* $x = L_x(q) = L_x(c)$, $q \in \boldsymbol{Q}$, $c \in \boldsymbol{C}$, $i \in L_\Gamma(c)$, $t_i \in L_\Gamma(q)$, $x_0 = L_X(q_0)$ *and* $q_0 \in \boldsymbol{Q}_0$, *for all* $i \in \mathbb{N}$ *and* $\bigcup_{i \geq 0} t_i = \mathbb{R}^+$

Note: Practically, discretization is based on choosing time steps $\triangle t$, which can be varied or fixed and can be chosen in domains other than $\mathbb{N}$ (e.g. values of $\triangle t$ are in $\mathbb{Q}$ or rounded $\mathbb{R}$)

4 Abstract Interpretation

Given the concrete $D^\flat$ and abstract $D^\sharp$ semantics domains of the system under analysis. A soundness relation σ is used to reason about the correspondence between a concrete and abstract semantics. A soundness relation can be formulated as $\sigma \in \wp(D^\flat \times D^\sharp)$. We say there exists an abstract approximation if we assume that for every concrete semantics we have an abstract approximation: $\forall s \in D^\flat . \exists t \in D^\sharp : (s, t) \in \sigma$. More closely, to ensure the soundness of the methodology, we use Galois connection between $D^\flat$ and $D^\sharp$. By associating partial order relation with the semantics domain (i.e. ($\mathrm{D}^\flat$, $\sqsubseteq$), ($\mathrm{D}^\sharp$, $\preceq$) which are partial order set), we say (α, γ) is a Galois connection between $D^\flat$ and $D^\sharp$, iff $\alpha : \mathrm{D}^\flat \rightarrow \mathrm{D}^\sharp$, $\gamma : \mathrm{D}^\sharp \rightarrow 2\mathrm{D}^\flat$ *and*$\forall s \in \mathrm{D}^\flat$, $\mathrm{t} \in \mathrm{D}^\sharp$, $\alpha(s) \preceq t \Leftrightarrow s \sqsubseteq \gamma(t)$. To build an abstract state space $S^\sharp$, which is an overapproximation of the concrete state space $S^\flat$, that is $\forall s \in S^\flat . \exists t \in S^\sharp . t = \alpha(s)$. As we build our abstraction, it is essential that all transitions of the concrete system are preserved in the abstract, but the concretization of abstract transitions does not result in spurious transitions.

Note. Imposing the existence of Galois connection between concrete and abstract domains is a tight requirement as sometimes it is not easy of even impossible to find the abstract α function. Cousot [8], proposed relaxing this requirement by only working with concretization function like in the case of Interval domain as shown below.

4.1 Abstraction

Definition 8. *Let CS be a continuous dynamical system and DS be a discrete transition system with configurations $\boldsymbol{Q}$ (with domain $\mathbb{X}$) and $\boldsymbol{A}$ (with domain $\mathbb{A}$) respectively. We say DS is an abstraction for CS, if there exists a concretization mapping $\gamma : \boldsymbol{A} \rightarrow 2^{\boldsymbol{Q}}$, such that :*

- *$A = (a, \tau)$ is a configuration where $a \in \mathbb{A}$ and τ is the set of time intervals, such that every interval is an increasing sequence of time steps during which the state is not varied.*
- *$\mathbb{A}_0 = \{a \in \mathbb{A} | \exists x \in \gamma(a) \wedge x \in \mathbb{X}_0\}$, $A_0 = (a, \tau)$ and $t_0 \in \tau$ where t_0 is the singular interval.*
- *For every continuous evolution Φ, if π is a sufficiently discretization of $\alpha(\Phi)$, then π is a discrete trace of DS.*
- *$\sigma \subseteq \mathbb{A} \times \mathbb{A}$ is a relation capturing abstract transitions; $\{a \rightarrow' a' | \exists x \in \gamma(a),\ t \in \mathbb{R} : x' = \Phi_x(t) \in \gamma(a') \wedge x \rightarrow x'\}$*

Lemma 1. *For a concrete transition system with transition relation $\rightarrow$ and a corresponding abstract transition system with transition relation $\rightarrow'$, we have $\rightarrow \subseteq \gamma(\rightarrow')$*

The set of successor states of $a \in \mathbb{A}$ is $Post(a) = \{a' \in \mathbb{A} | a \rightarrow' a'\}$ and the set of reachable states *Reach* of a transition system can then be defined as follow:

Definition 9. *For a transition system, the set of reachable states $AReach \subseteq \boldsymbol{A}$ is defined as:*

- *$AReach^{(0)} := \boldsymbol{A}_0$*
- *$AReach^{inc(i)} := Post(AReach^{(i)})$, $\forall i > 0$*
- *$AReach := \bigcup_{i \geq 0} AReach^{(i)}$*

The verification problem is stated as follows: Given a transition system DS and a set of (un)safe states $B \subseteq \mathbf{A}$, is there a trace starting from $\mathbf{A}_0$ that can reach B; whether $AReach \cap B$ is empty or not. We say that AReach is an over approximation of Reach

Lemma 2. *Let CS be a continuous dynamical system and DS a discrete transition system as its abstraction, we have: $Reach \subseteq \{q \in \boldsymbol{Q} | \exists a \in AReach \wedge q \in \gamma(a)\}$*

4.2 Interval Abstract Domain

In this paper, we choose intervals as the numerical abstract domains with which we compute the abstract semantics of continuous systems. We will briefly outline constructing the abstract domain by following the modular approach presented by Mine in [9] were he propose starting from a basis representing abstraction of state variables as well as basic operations and using lifting of the concretization and abstraction functions to sets and functions representing expressions and

transfer functions. The interval abstract domain D^ι is based on the classical concept of interval arithmetics [11] and was adapted by Cousot and Cousot in [7].

The relation between the concrete and abstract domain can be described by a partial Galois connection $(\wp(\mathbb{R}), \leq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (B, \sqsubseteq_B^\sharp)$ with:

Interval Basis B^ι. B is a set of intevals with bounds in $\mathbb{R}$, where B : $\{\bot_B^\iota\} \cup \{[a,b] | a \in \mathbb{R} \cup \{-\infty\}, b \in \mathbb{R} \cup \{\infty\}, a \leq b\}$. We say B^ι contains $[-\infty, \infty]$ which denote the whole set $\mathbb{R}$, the singletons $[a,a]$, when $a \in \mathbb{R}$, and the empty interval denoted by $\bot_B^\iota$.

Interval Basis Structure The partial order $\sqsubseteq_B^\iota$ is defined as $[a,b] \sqsubseteq_B^\iota [a',b'] \triangleq a \geq a'$ and $b \leq b'$ and $\bot_B^\iota$ is the smallest element. The basis concretization γ_B^ι is defined as γ_B^ι ([a,b]) $\triangleq \{x \in \mathbb{R} | a \leq x \leq b\} and \gamma_B^\iota(\bot_B^\iota) \triangleq 0$.

Interval Basis Operators. We define the required operators for a basis the following way:

– $\cup_B^\iota$ and $\cup_B^\iota$:

$$X^\sharp \cup_B^\iota Y^\sharp \triangleq \begin{cases} [min(a,a'),\ max(b,b')] & if\ X^\sharp = [a,b] \text{and} Y^\sharp = [a',b'] \\ X^\sharp & if Y^\sharp = \bot_B^\iota \\ Y^\sharp & if X^\sharp = \bot_B^\iota \end{cases}$$

$$X^\sharp \cap_B^\iota Y^\sharp \triangleq \begin{cases} [max(a,a'),\ min(b,b')] & if\ X^\sharp = [a,b]\ and\ Y^\sharp = [a',b'] \\ & and\ max(a,a') \ \leq \ min(b,b') \\ \bot_B^\iota & otherwise \end{cases}$$

– In general, abstraction for arithmetics operators can be described as follows:
- $\neg^\sharp : B \to B$ such that$\gamma_B^\iota(\neg^\sharp X^\sharp) \supseteq \{\neg x | x \in \gamma_B^\iota(X^\sharp)\}$
- $\diamondsuit^\sharp : B^2 \to B$ such that$\gamma_B^\iota(X^\sharp\ \diamondsuit^\sharp Y^\sharp) \supseteq \{x \diamondsuit y | x \in \gamma_B^\iota(X^\sharp$, y $\in \gamma_B^\iota(Y^\sharp)\}$ and $\diamondsuit \in \{+, -, \times, \div\}$

For interval arithmetics we have the following:

[a,b]$^\iota$	$\triangleq$	[a+b]	
[a,b] $+^\iota [a',b']$	$\triangleq$	[a+a', b+b']	
[a,b] $-^\iota [a',b']$	$\triangleq$	[a-b', b-a']	
[a,b] $\times^\iota [a',b']$	$\triangleq$	[min (aa', ab', ba', bb'), max (aa', ab', ba', bb')]	
$1 \div [a,b]$	$\triangleq$	$[1 \div b, 1 \div a]$	$if\ 0 \notin [a,b]$
[a, b] $\div [a',b']$	$\triangleq$	[a,b] $\times 1 \div [a',b']$	

Note We use α^ι and γ^ι also to denote liftings of the functions α_B^ι and γ_B^ι to sets, relations and functions (e.g, expression evaluation, variables assignments, etc.).

Interval domain $\mathbb{I}$ is a conservative domain which overapproximate the original one $\mathbb{R}$. We can describe the discrete interval transition systems as following:

Definition 10. *A discrete interval transition system is a discrete transition system* $T_c = (A, A_0, \sigma, L)$ *with domain* $\mathbb{I}^n$*, where* n *is the dimension of the state space, where:*

- $I = (a, \tau)$ *is a configuration where* $a \in \mathbb{I}$ *and* τ *is the set of time intervals, such that every interval is an increasing sequence of time steps during which the state is not varied.*
- $\mathbb{I}_0 = \{a \in \mathbb{I} | \exists x \in \gamma(a) \wedge x \in \mathbb{X}_0\}$*,* $I_0 = (a, \tau)$ *and* $t_0 \in \tau$ *where* t_0 *is the initial singular interval.*
- $\sigma \subseteq \mathbb{I} \times \mathbb{I}$ *is a relation capturing abstract transitions;* $\{a \rightarrow a' | \exists x \in \gamma(a),\ t \in \mathbb{R} \ :\ x' = \Phi_x(t) \in \gamma(a') \wedge x \rightarrow x'\}$

We can therefore deduce the following theorem:

Theorem 1. *Let CS be a continuous system and DS be a discrete Interval Transition system, as defined above. Then DS is an abstraction for CS.*

5 Application: Tunnel Diode Oscillator

Several interval arithmetic implementation for the the initial value problem have been developed, see for instance [12] for an overview. We have used in this report, AWA tool developed by Lohner [10], it has the advantage of efficiently dealing with the wrapping effect (error resulting from enclosing non rectangular regions by rectangular ones, which can lead to exponential growth in overapproximation, hence reducing the precision). To illustrate our methodology, we used the tunnel diode oscillator (see Figure2).

Tunnel diode oscillator circuit has attracted the interest of many researchers working on the verification of AMS designs. See for instance [27, 23, 15]. This is largely due to its wide usage in analog system designs (i.e.,the most common application of a tunnel diode is in high-frequency oscillator circuits) and technically because of its non linear behavior from which several properties can be deduced. Tunnel diodes exploit a phenomenon called resonant tunneling to provide interesting forward-bias characteristics, due to its negative resistance characteristic at very low forward bias voltages. That means that for some range of voltages, the current decreases with increasing voltage. This is in contrast with conventional diodes have a nonlinear I-V characteristic, but the slope of the curve is always positive. This characteristic makes the tunnel diode useful as oscillator. When a small forward-bias voltage is applied across a tunnel diode, it begins to conduct current. As the voltage is increased, the current increases and reaches a peak value called the peak current. If the voltage is increased a little more, the current actually begins to decrease until it reaches a low point called the valley current. If the voltage is increased further yet, the current begins to increase again, this time without decreasing into another valley.

In this section, we present results from experiments with the tunnel-diode oscillator circuit. We focus on the current I_L and the voltage V_C across the

tunnel diode in parallel with the capacitor of a serial RLC circuit (see Figure 2). The state equations of the circuits are given as follows:

$$\dot{V}_C = \frac{1}{x}(-I_d(V_C) + I_L) \text{ and } \dot{I} = \frac{1}{L}(-V_C - \frac{1}{G}I_L + V_{in})$$

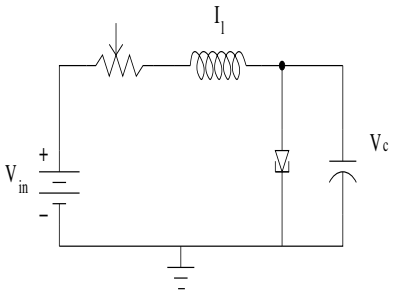


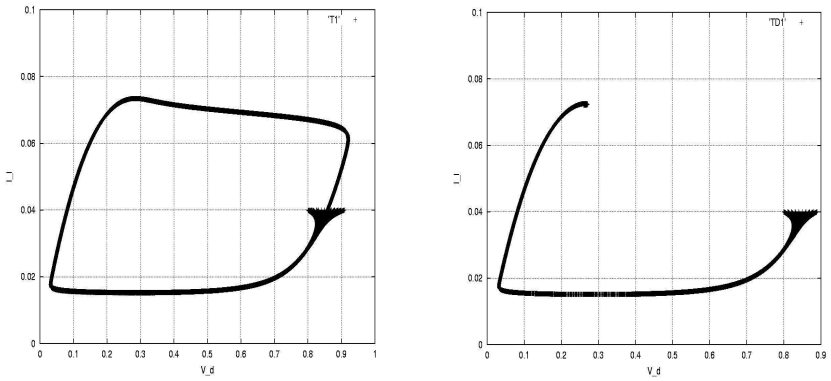
Fig. 2. Oscillation

Where $I_d(V_C)$ describes the non-linear tunnel diode behavior. We analyze the circuits in two modes. The first when the circuit is in stable oscillation for a given set of parameters, the other case when this oscillation dies out. The kind of properties we are interested to verify can be for example: *The system behavior will be the same for the set of initial condition*, or *For which set of parameters values, circuit oscillates?*. In this paper, we limits ourself with properties of the first type, similar to the one verified in [23, 15]. We use a variant of ACTL temporal logic extended with predicates of Real and time intervals restricting temporal operators, see [13] for more details.

We chose these two different set of parameters values of the oscillator circuit $\{C = 1000e^{-12},\ L = 1e^{-6},\ G = 5000e^{-3},\ Vin = 0.3\}$ and $\{C = 1000e^{-12},\ L = 1e^{-6},\ G = 2000e^{-3},\ Vin = 0.3\}$ along with the set of initial values of voltages $[0.8\ V,\ 0.9\ V]$ and currents $0.04\ mA$ and the analysis region of interest $-1\ V \leq V_C \leq 1\ V$ and $0.01\ mA \leq I_L \leq 0.9\ mA$. By using interval based analysis, with the first set of parameters, we can find out that the circuit is oscillating for the given set of initial conditions (see Figure 3.a). By applying the reachability analysis in Definition 9, we verify that for the given initial conditions, the trajectory will be within the analysis region. Suppose we want to verify the following property on the set of trajectories[23]:

$$\forall\Box_{[0,1e^{-6}]}(\forall\Diamond(I_L \leq 0.02)) \ \wedge \ \forall\Box_{[0,1e^{-6}]}(\forall\Diamond(I_L \geq 0.06))$$

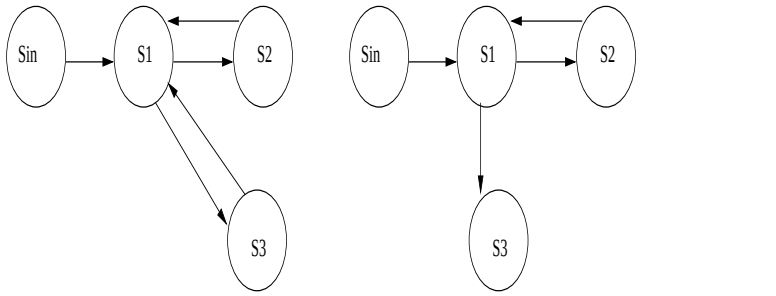
Which can be understood as within the time interval $[0, 1e^{-6}]$ on every computation path, first predicate is true at some future time as well as for the second predicate. This property checks for oscillation behavior. We can divide the state space into 3 states, each corresponding to a set of predicate inequalities extracted



(a. Oscillation) (b. No Oscillation)

Fig. 3. Oscillation

from the monitors in addition to the initial condition (as shown in figure 4.a). It is abvious that within the specified time interval, the property in verified.



(a. Oscillation TS) (b. Non Oscillating TS)

Fig. 4. Transition Systems

Which can be understood as within the time interval $[0, 1e^{-6}]$ on every computation path, first predicate is true at some future time as well as for the second predicate. This property checks for oscillation behavior. We divide the state space into 3 states in addition to the initial condition, each corresponding to a set of predicate inequalities extracted from the formulas ($P_1 = I_L \leq 0.02$, $P_2 = 0.02 < I_L < 0.06$, $P_3 = \ I_L \geq 0.06$), as shown in Figure 4.a. We can use a labeling function *Prop* associating a set of atomic propositions to each state; $Prop : Q \rightarrow 2^{AP}$. For example, $Prop(S1) = \{\bar{P_1}, P_2, \bar{P_3}\}$. Monitors synthesized from the temporal logic can be used similar to threshold detection. Each time a condition is changed, monitors triggers a change of states of the abstract tran-

sition system. For example, there is a transition between state $S1$ and state $S2$, as the trajectories generated by interval analysis and monitored by a monitoring automata cross the region satisfying the conjunction of atomic propositions $\bar{P_1} \wedge P_2 \wedge \bar{P_3}$ to the region satisfying $P_1 \wedge \bar{P_2} \wedge \bar{P_3}$. In order, to verify oscillation condition using model checking on the abstract model , we translate CT-CTL properties to TCTL properties. The model checking procedure is then straight forward and model checking tools like SMV or Hytech can be used to check such properties.

By following the same procedure for the system with the second set of parameters, but with the same initial conditions, we can find out that the circuit is non oscillating. When the circuit starts up, the energy of the system is lost due to the positive circuit resistance. Starting from any point in the analysis region, the oscillations die down to the equilibrium point (see Figure 3.b). By applying the reachability analysis in Definition 9, we verify, however, that for the given initial conditions, the trajectory will be within the analysis region. If we want to verify the oscillation property, we start by creating the abstract transition system as shown in Figure 4.b. and by applying model checking algorithms, we find out that the condition is not holding. This can be inferred by looking at the transition system in Figure 4.b, as there is no transition out from $S3$, which can be interpreted as once the system reaches $S3$, it remains deadlocked there during the verification period.

6 Related Work

An important issue of algorithmic methods in formal verification and model checking of AMS circuits are the solution of continuous systems; that is, the collection of continuous time trajectories starting from a set of initial continuous states where in practice the initial conditions are usually not known exactly but only known to lie within some range. Several methods for approximating reachable sets for continuous dynamics have been proposed. These methods rely on the discretization of the continuous state space. Among the most important approximations are those based on cubical and polyhedral representation. This approaches were pursued by Kurshan and McMillan in [19] and Greenstreet [20, 21] respectively were they proposed verifying digital properties at the transistor level. A variant approach of polyhedral based analysis was adapted by Dang and Maler [1] and implemented in the tool d/dt and by Chutinan and Krogh and implemented in their tool Checkmate [2] which supports in addition temporal verification. In [22], Dang *et. al*, Dang and Maler, used d/dt for the verification of analog systems described with differential algebraic equations (DAEs) and apply it to the verification of a biquad low-pass filter. In addition they proposed using techniques from optimal control (i.e hybrid constrained optimization) in order to find bounds of the reachability and they applied this technique to the verification of first order $\Delta - \Sigma$ modulator. In [23], the authors used Checkmate tool for the verification of AMS designs, i.e, tunnel diode oscillator and $\Delta - \Sigma$ modulator

In [15], Hartong *et. al* proposed discretizing the whole state space into variable sized regions to represent the state space and he used some kind of estimation techniques to describe possible transitions between partitions. The discretized state space is then encoded and CTL based model checking is applied. The paper gave no proof of soundness and It is not clear how the whole CTL is supported as in general approximation can only preserves a subset namely the ACTL. The proposed approach was implemented in a tool called Amcheck, developed at University of Hannover. in [16], they extended their methodology for the verification of time properties (i.e rise and fall time) of analog circuits.

Several abstraction techniques have been proposed for continuous and Hybrid systems. One of the early work for applying abstract interpretation to Hybrid systems was proposed by Halbwachs *et.al* [28] as extension to a previous work with Cousot [14]. In this work, convex approximation of linear equations is described. A variant of this work is latter implemented in HyTech [3]. The main limitation of this approach is the applicability only for linear systems, which practically restrict the class of systems under verification. Hypertech [4], is an extended version of Hytech, where they add an interval library to support verification of non linear systems. The idea presented is similar to our proposal, however we diverge from them in that we propose combining interval analysis with property guided abstraction which can lead to efficient analysis of the non linear systems. Henzinger *et. al* [5] present a methodology for algorithmically analyzing nonlinear hybrid systems by first translating the system to linear hybrid automata, and then using automated model-checking tools. In a linear hybrid automaton, the continuous environment is partitioned into a finite number of classes such that within each class, the continuous variables are governed by a constant polyhedral differential inclusions. Although, the idea is very interesting, generated linear hybrid automata are still large enough to be easily model checked. One other problem of this approach is the linearization of the dynamics which results in losing information which might be of importance like the effect of external disturbance for example, make it impractical for the verification of analog circuits.

Predicate abstractions [30] have been successful in the verification of infinite systems. I have been extended to the abstraction of verification of hybrid systems with different complexity, see for instance the work by Alur *et. al* [6], Ahish *et. al* [31]. One main problem in this approach is in choosing the correct predicate. In our proposed methodology, we plan to use predicates in temporal logic as a mean to build the abstract state space, by monitoring the execution, each set of solutions satisfying certain predicates are represented by a certain discrete state and finally, Clarke *et. al* [32], extended the Checkmate verification toolbox with an abstraction refinement methodology.

Program monitoring have been applied sucessffuly for hardware and software analysis. Motivated with the success of PSL assertion languages for hardware verification, Maler *et. al.* [26] proposed a simple methodology for monitoring the simulation of continuous signals by extending the PSL logic to support predicates over the reals (signals) , the goal is to verify continuous and analog systems.

Monitoring was applied within the Charon design framework [25] where timed automata and linear hybrid automaton can be used to monitor real-time and hybrid behavior. Recently similar ideas using hybrid automata as monitors have been integrated with the PHVAR a hybrid system analysis tool [27] that provides sound verification results based on linear hybrid automata approximations and was used to verify properties of piecewise models of oscillator like amplitude bounds and phase jitter.

Interval and affine arithmetics have been used in the analysis of analog circuits like in [17], were the authors presents an approach for equivalence checking of linear analog circuits with parameter tolerances. The goal is to prove that an actual circuit fulfills a specification in a given frequency interval for all parameter variations and in [18], affine arithmetics was used to for helping in the circuit sizing.

7 Conclusion

The lack of methods for computing reachable sets of continuous dynamics has been the main obstacle towards an algorithmic verification methodology for hybrid systems. This motivated us to tackle first the reachability problem of continuous systems. Unlike the conventional approaches which attempt to find exact solutions and are thus limited by undecidability of most non-trivial systems, our approach is based on an efficient method for abstracting the continuous behavior using combination of techniques from numerical methods, abstract interpretation, and program monitoring.

Future Work. The description of the methodology presented in this section was general. Different issues will be raised throughout the development. We present below some issues we think are of important interests:

- Extends the methodology for other domains like affine arithmetics which have been developed to enhance precision.
- Explore more complex case studies; currently we are working on the verification of phase locked loop designs and scmitt trigger.
- Extends the methodology for Hybrid systems.

References

1. Eugene Asarin, Thao Dang, Oded Maler: The d/dt Tool for Verification of Hybrid Systems. in Computer Aided Verification, LNCS 2404, Springer, 2002, pp. 365-370.
2. A. Chutinan and B. H. Krogh, Computational techniques for hybrid system verification. IEEE Trans. on Automatic Control, vol. 48, no. 1, 2003, pp. 64-75.
3. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A Model Checker for Hybrid Systems. Software Tools for Technology Transfer 1:110-122, 1997.
4. Thomas A. Henzinger, Benjamin Horowitz, Rupak Majumdar, Howard Wong-Toi: Beyond HYTECH: Hybrid Systems Analysis Using Interval Numerical Methods. HSCC 2000: 130-144

5. Thomas A. Henzinger, Pei-Hsin Ho: Algorithmic Analysis of Nonlinear Hybrid Systems. CAV 1995: 225-238
6. R. Alur, T. Dang, and F. Ivancic. Counter-example guided predicate abstraction of hybrid systems, Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2003
7. P. Cousot, R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 238252, Los Angeles, California, 1977. ACM Press, New York, NY, USA.
8. Patrick Cousot, Radhia Cousot: Abstract Interpretation Frameworks. J. Log. Comput. 2(4): 511-547 (1992)
9. A. Mine, Weakly Relational Numerical Abstract Domains. PhD thesis Report, Computer Science Department, the Ecole Normal Suprieure , France, Dec. 2004
10. R. J. Lohner, Enclosing the solutions of ordinary initial and boundary value problems, in Computer Arithmetic: Scientific Computation and Programming Language, Wiley-Teubner Series in Computer Science, Stuttgart, 1987, 255-286
11. R. E. Moore Methods and Applications of Interval Analysis, Society for Industrial Applied Mathematics, Philadelphia, 1979, ISBN 0898711614.
12. R. B. Kearfott. Interval computations: Introduction, uses, and resources. Euromath Bulletin, 2(1):95-112, 1996.
13. M. Zaki, S. Tahar, G. Bois, Formal Verification of Analog and Mixed Signal Systems. In preparation
14. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In Proceedings of the 5th Annual ACM Symposium on Principles of Programming, pages 84–97, 1978.
15. W. Hartong, R. Klausen, L. Hedrich, "Formal Verification for Nonlinear Analog Systems: Approaches to Model and Equivalence Checking," Advanced Formal Verification, R. Drechsler, ed., Kluwer Academic Publishers, Boston, January 2004, pp. 205-245.
16. D. Grabowski, D. Platte, L. Hedrich, and E. Barke, Time Constrained Verification of Analog Circuits using Model-Checking Algorithms,Workshop on Formal Verification of Analog Circuits, 2005.
17. L. Hedrich and E. Barke,A Formal Approach to Verification of Linear Analog Circuits with Parameter Tolerances. Design, Automation and Test in Europe, 1998, pp. 649-654.
18. Andreas C. Lemke, Lars Hedrich, Erich Barke: Analog circuit sizing based on formal methods using affine arithmetic. ICCAD 2002: 486-489
19. R.P. Kurshan and K.L. McMillan. Analysis of digital circuits through symbolic reduction. IEEE Trans. on Computer-Aided Design 10:13501371, 1991.
20. Mark R. Greenstreet, Ian Mitchell: Reachability Analysis Using Polygonal Projections. HSCC 1999: 103-116
21. Mark R. Greenstreet, Ian Mitchell: Integrating Projections. HSCC 1998: 159-174
22. Thao Dang, Alexandre Donze, O.M.: Verification of analog and mixed-signal circuits using hybrid system techniques. In: Formal Methods in Computer-Aided Design (FMCAD 2004), Austin, Texas, November 14-17, 2004. (2004)
23. Gupta, S.; Krogh, B.H.; Rutenbar, R.A.:Towards formal verification of analog designs, IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004. , 7-11 Nov. 2004 Page(s):210 - 217

24. A. Bemporad and M. Morari Verification of hybrid systems via mathematical programming.In F.W. Vaandrager and J.H. van Schuppen, editors, Hybrid Systems: Computation and Control, volume 1569 of Lecture Notes in Computer Science, pages 31–45. Springer Verlag, 1999.
25. Li Tan, Jesung Kim, Insup Lee: Testing and Monitoring Model-based Generated Program. Electr. Notes Theor. Comput. Sci. 89(2): (2003)
26. Oded Maler, Dejan Nickovic: Monitoring Temporal Properties of Continuous Signals. FORMATS/FTRTFT 2004: 152-166
27. G. Frehse, B. Krogh, R. Rutenbar, O. Maler Time Domain Verification of Oscillator Circuit Properties, Workshop on Formal Verification of Analog Circuits, 2005
28. N. Halbwachs, P. Raymond, and Y. Proy. Verification of linear hybrid systems by means of convex approximations. In B. LeCharlier, editor, Proceedings of International Symposium on Static Analysis, volume 864 of Lecture Notes in Computer Science. Springer-Verlag, September 1994.
29. T. A. Henzinger and P. Ho. A note on abstract-interpretation strategies for hybrid automata. In Hybrid Systems II, Lecture Notes in Computer Science 999, Springer-Verlag, 1995, pp. 252-264.
30. Susanne Graf, Hassen Sai"di: Construction of Abstract State Graphs with PVS. CAV 1997: 72-83
31. Ashish Tiwari, Gaurav Khanna: Series of Abstractions for Hybrid Automata. HSCC 2002: 465-478
32. Edmund M. Clarke, Ansgar Fehnker, Zhi Han, Bruce H. Krogh, Olaf Stursberg, Michael Theobald: Verification of Hybrid Systems Based on Counterexample-Guided Abstraction Refinement. TACAS 2003: 192-207

Formal Verification of Spacing Properties of an Air Traffic Management Concept*

César A. Muñoz† Gilles Dowek‡

August 3, 2005

Abstract

We propose a mathematical model to verify continuous safety properties of the NASA's Small Aircraft Transportation–Higher Volume Operations (SATS–HVO) concept. The technique described in this paper allows for an exhaustive generation of nominal aircraft operations and mechanical verification of spacing properties, which are critical to the concept safety. Additionally, the formal model yields analytical formulas to compute spacing minima on arrival approaches. These formulas, which are parameterized by the geometry of the SATS–HVO terminal area and the performance of the aircraft, can be used to configure a baseline procedure for self separation. The mathematical development presented in this paper has been formally verified in the Prototype Verification System (PVS).

Acronyms

AMM	Airport Management Module
ATM	Air Traffic Management
CONOPS	Concept for Nominal Operations
HVO	Higher Volume Operations
IAF	Initial Approach Fix
IMC	Instrument Meteorological Conditions
MAHF	Missed Approach Holding Fix
PVS	Prototype Verification System
SATS	Small Aircraft Transportation System
SCA	Self Controlled Area

1 Introduction

The safety objective of an Air Traffic Management (ATM) concept is to keep aircraft sufficiently separated. This is accomplished trough a set of operational procedures, on-board equipment, and aerospace restrictions. All these elements form the concept for nominal operations (CONOPS). The safety assumption of an ATM system is that aircraft flying in compliance with the CONOPS are separated from each other.

Emerging and more reliable surveillance and communication technologies enable air-ground distributed ATM concepts where pilots and air traffic controllers share the responsibility for traffic separation. One of such concepts is the NASA's *Small Aircraft Transportation System (SATS)* [5]. The SATS program aims to increase access to small airports in the US during instrument approach operations. The concept of operation, which is known as Higher Volume Operation (SATS–HVO) [1], consists of four components:

*This work was supported by the National Aeronautics and Space Administration under NASA Cooperative Agreement NCC-1-02043.

†National Institute of Aerospace (NIA), 100 Exploration Way, Hampton, VA 23666, USA. Email: munoz@nianet.org.

‡Laboratoire d'Informatique (LIX), École polytechnique, 91128 Palaiseau Cedex, France. Email: Gilles.Dowek@polytechnique.fr.

(1) a designated airspace surrounding the airport called the Self Controlled Area (SCA); (2) a centralized automated system called the Airport Management Module (AMM); (3) aircraft to aircraft and aircraft to AMM data communication; and, (4) distributed on-board navigation tools.

A key aspect of the SATS–HVO concept is that, under nominal operations, aircraft in the SCA are self separated, i.e., pilots are responsible for separation without assistance of an air traffic controller. Self separation is achieved by a baseline procedure where an aircraft in a holding fix delays its final approach initiation until it meets a spacing safety threshold with respect to its lead aircraft. The threshold guarantees a minimum separation until the aircraft lands or leaves the SCA. In this paper, the term *spacing* refers to linear separation of an aircraft with respect to its lead aircraft in the landing/departing sequence.

The task of showing that the design of the SATS–HVO concept is correct is accomplished using formal mathematical analysis. A high level mathematical model of the concept is described in [2]. The mathematical model is a conservative abstraction of the real concept. It considers all type of aircraft performances, including some that may not be physically possible. This conservative abstraction yields a discrete finite transition system that enables exhaustive state exploration of nominal operations. The model was mechanically checked [4] in the verification system PVS [6]. For example, it has been formally verified that the SATS–HVO concept allows up to four concurrent arrival operations in the SCA, which is better than the current one-in one-out restriction for this type of operations, and that eventually all arrival aircraft land, i.e., there are no deadlocks.

The discrete model does not support verification of spacing properties as these properties require a more precise modeling of the geometry of the SCA and of the aircraft performance parameters. This papers proposes a hybrid model that extends the discrete model presented in [2]. In contrast to the original model, the proposed model enables the verification of continuous safety properties of nominal SATS–HVO operations. To this end, aircraft performances, such as ground speed ranges, and information about the SCA geometry, such as length of the approach segments, are integrated into the original model. Thus, in the new model, the concept of operations is described by the *continuous* dynamics of aircraft and the *discrete* events within the SCA.

2 SATS–HVO CONOPS

The SATS–HVO concept of operations [1] is a collection of rules and procedures which, when followed, will provide separation assurance during transition to the SCA, approach, missed approach, landing, takeoff, departure, and transition out of the SCA. On board navigation tools will provide advisories to aid pilots in following these procedures. The other components of the concept are the Self Controlled Area (SCA) and the Airport Management Module (AMM).

The SCA is an airspace volume surrounding the airport facility. The design of the SCA is similar to a GPS T approach [3], were pilots are required to fly by latitude/longitude points in the space, called *fixes*, in order to perform a landing approach or a departure. In a T approach, the fixes are geographically arranged as a T. Some of these fixes are *holding fixes*. Under particular circumstances, an aircraft is allowed to fly around a holding fix waiting for another aircraft to go first in a landing approach. A *missed approach holding fix* is a holding fix to which an aircraft will proceed in case it executes a missed approach.

Figure 1 shows a top view of a nominal SCA. The fixes are the right and left initial arrival fixes (IAF-R, IAF-L), intermediate fix (IF), final approach fix (FAF), and right and left departure fixes (DF-R, DF-L).[1] Moreover, there are right and left missed approach holding fixes (MAHF-R, MAHF-L) at 2000 feet and at 3000 feet.

The AMM is an automated system which will typically reside at the airport grounds. It serves as an arbiter and sequencer of the SCA. It receives state information from aircraft in the vicinity of the airport and communicates with aircraft via data link. The AMM minimally supports flight operations by implementing entry rules, providing follow notifications, and assigning missed approach holding fixes.

[1] As it is usually depicted, right and left are relative to the pilot facing the runway, i.e., opposite from the reader point of view.

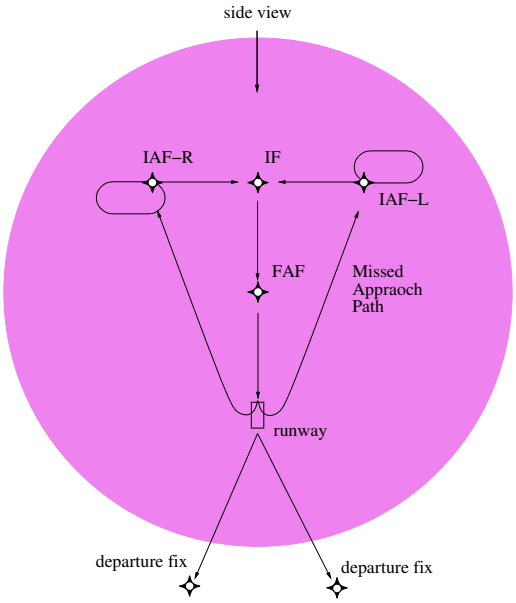


Figure 1: Top view of SCA

There are two types of entry into the SCA: *vertical entry* and *lateral entry*. In a vertical entry, an aircraft flies to the IAF at an altitude above the SCA. The aircraft holds at the IAF above the SCA until entry is granted by the AMM. The aircraft then descends over the IAF flying a race track trajectory through 4000 to the 3000 feet holding fix. A lateral entry is possible when there are no aircraft at or assigned to the IAF. In this case, the aircraft proceeds to the IAF in a flight trajectory to arrive at the IAF at or above 2000 feet. When an entry is granted by the AMM, the aircraft receives a *follow notification* and a *missed approach holding fix assignment*. The follow notification is either *none*, if it is the first aircraft in the landing sequence, or the identification of a *lead* aircraft. An aircraft should proceed from the IAF to the IF, and from there to the FAF and, finally, to the runway threshold, soon after some spacing criteria with respect to the lead aircraft are satisfied. In case of a missed approach, the aircraft flies to its assigned missed approach holding fix at the lowest available altitude (2000 or 3000 feet). Then, it re-initiates the approach and either follows a normal landing procedure or leaves the SCA. Missed approaches are assigned by the AMM in an alternating basis. This technique ensures that consecutive aircraft on missed approach are not flying to the same missed approach holding fix.

For the analysis presented in this paper, we only consider arrival operations. This simplification does not affect the result of the formal verification as arriving aircraft are geographically separated from departing aircraft.

3 Discrete Model

The discrete model of the SATS–HVO concept is a non-deterministic, asynchronous, state transition system that describes nominal operations in the SCA. We refer to [2] for a detailed description of the model.

The global discrete state of the system is composed by the state of the AMM and the sate of the SCA. The state of the AMM includes the next available landing sequence and the next alternating missed approach holding fix. Landing sequences are natural numbers (starting from 1), missed approach holding fixes are either *left* or *right*. The SCA is logically divided into 12 zones (see Figure 2):

- `holding3(`*left*`)` and `holding3(`*right*`)`: Left and right holding patterns at 3000 feet.

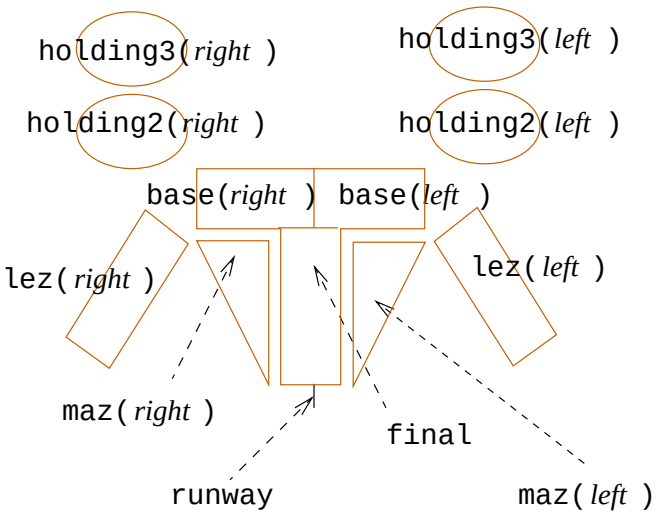


Figure 2: SCA 12 zones

- `holding2(`*left*`)` and `holding2(`*right*`)`: Left and right holding patterns at 2000 feet.
- `lez(`*left*`)` and `lez(`*right*`)`: Left and right lateral entry zones.
- `base(`*left*`)` and `base(`*right*`)`: Left and right base segments.
- `maz(`*left*`)` and `maz(`*right*`)`: Left and right missed approach zones.
- `final` and `runway`: Final segment and runway.

The state of each of these zones is a list of aircraft states. The state of an aircraft is a record with 2 fields: landing sequence and MAHF assignment. Aircraft identifications are implicit in this model. The lists of aircraft states define time/space relations between aircraft. In particular, the order of aircraft in a list is the order of arrival to the zone. Furthermore, zones behave as first-in first-out data structures: aircraft are removed from the head of one zone and added to the tail of the next zone.

Operational rules are modeled as transitions over the global state of the system. Seventeen transitions were identified: Vertical entry (left and right), Lateral entry (left and right), Descend from 3000 to 2000 feet (left and right), Approach initiation for vertical entry (left and right), Approach initiation for lateral entry (left and right), Transition from base segment to final segment (left and right), Landing, Taxiing, Missed approach initiation, and Determination of lowest available altitude (left and right).

It has been shown that the concept satisfies, among many others, the following high-level safety properties:

- There are at most four concurrent arrival operations within the SCA.
- There is at most one aircraft at each initial approach fix at a given holding altitude.
- There are no more than two aircraft on missed approach flying to the same missed approach holding fix.
- There is always a free missed approach holding fix for any aircraft on missed approach.
- Aircraft land according to their landing sequence.
- Arriving and departing aircraft eventually land or leave the SCA.

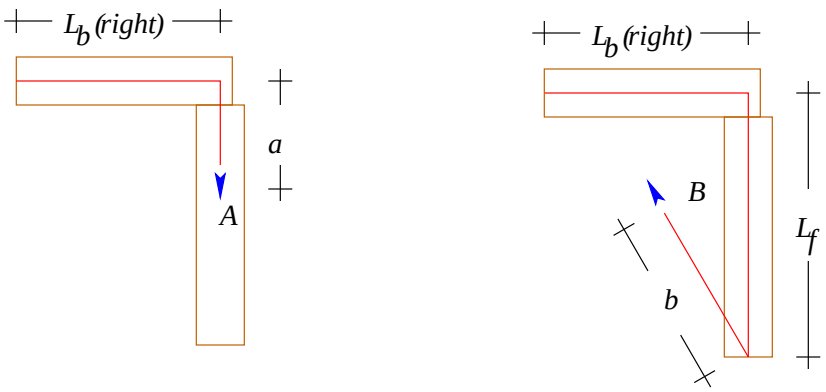


Figure 3: Linear distance from IAF

4 The Spacing Problem, Formally

The geometry of the SCA is described by $L_b(left)$, $L_b(right)$, L_f, $L_m(left)$, and $L_m(right)$, which stand for the linear lengths of the left and right base segments, final segment, and left and right missed approach zones, respectively.

The position of an aircraft on final approach or missed approach is defined as the linear distance from its IAF. We use $D_A(t)$ to denote the linear distance at time t of the aircraft A from its IAF. For example, positions of aircraft A and B in Figure 3 are given by $D_A(t) = L_b(right) + a$ and $D_B(t) = L_b(right) + L_f + b$.

The time when an aircraft A initiates the final approach, i.e., when it enters the base segment, is denoted T_A. Hence,

$$D_A(T_A) \quad = \quad 0. \tag{1}$$

If A is on final approach or missed approach at time t:

$$T_A \ \leq \ t. \tag{2}$$

Let iaf_A be the initial approach fix of A, and $mahf_A$ its missed approach holding fix assignment.

- If A is on the base segment:

$$0 \ \leq \ D_A(t) \ \leq \ L_b(iaf_A). \tag{3}$$

- If A is on the final segment:

$$L_b(iaf_A) \ \leq \ D_A(t) \ \leq \ L_b(iaf_A) + L_f. \tag{4}$$

- If A is on missed approach:

$$L_b(iaf_A) + L_f \ \leq \ D_A(t) \ \leq \ L_b(iaf_A) + L_f + L_m(mahf_A). \tag{5}$$

We assume that the speed of an aircraft may vary with time in the interval $[v_{\min}, v_{\max}]$. Therefore, for an aircraft A on final approach or missed approach at time t,

$$(t_1 - t_0)v_{\min} \leq \ D_A(t_1) - D_A(t_0) \ \leq \ (t_1 - t_0)v_{\max}, \tag{6}$$

if $T_A \leq t_0 \leq t_1 \leq t$.

According to the concept of operations, the following properties also hold.

- Let B be the lead aircraft of A. If B is on final approach when A initiates the approach:

$$\begin{aligned} T_B &\leq T_A, \quad \text{and} && (7)\\ S_0 + L_b(iaf_B) - L_b(iaf_A) &\leq D_B(T_A), && (8) \end{aligned}$$

  where S_0 is the initial procedural spacing that a trail aircraft has to meet in order to start its approach.

- If the aircraft B is on missed approach when A initiates the final approach:

$$L_b(iaf_B) + L_f \leq D_B(T_A). \tag{9}$$

If B is before A in the landing sequence, the spacing between A and B is defined as

$$S_{A\to B}(t) \equiv D_B(t) - D_A(t) + L_b(iaf_A) - L_b(iaf_B). \tag{10}$$

Note that according to this definition, spacing of aircraft on opposite base segments is computed as if both aircraft were in the same segment. In particular, if the base segments have the same length, two aircraft on opposite IAFs have no spacing.

Propositions 1 and 2 specify the spacings properties we want to check.

Proposition 1. *Under nominal operations, aircraft on final approach are spaced to their lead aircraft. Formally, for any aircraft A and B on final approach at time t, such that B is the lead aircraft of A,*

$$S_T \leq S_{A\to B}(t).$$

Proposition 2. *Under nominal operations, aircraft on missed approach at the same fix are spaced. Formally, for any aircraft A and B on missed approach at the same fix at time t, such that B is before A in the landing sequence,*

$$S_{MAZ} \leq S_{A\to B}(t).$$

The constants S_T and S_{MAZ} are the theoretical spacing that the concept guarantees on final approach and missed approach, respectively. As we will see in Section 7, these constants are determined by the performance of the aircraft, the geometry of the SCA, and the baseline self separation procedure:

$$\begin{aligned} S_T &\equiv S_0 - (L_{max} + L_f - S_0)\Delta_v, \quad \text{and} && (11)\\ S_{MAZ} &\equiv \min(L_{min} + L_f - L_{maz}\Delta_v, 2S_0 - (L_{max} + L_f + L_{maz} - S_0)\Delta_v), && (12) \end{aligned}$$

where

$$\begin{aligned} L_{min} &\equiv \min(L_b(\mathit{left}), L_b(\mathit{right})), && (13)\\ L_{max} &\equiv \max(L_b(\mathit{left}), L_b(\mathit{right})), && (14)\\ L_{maz} &\equiv \max(L_m(\mathit{left}), L_b(\mathit{right})), \quad \text{and} && (15)\\ \Delta_v &\equiv \frac{v_{\max} - v_{\min}}{v_{\min}}. && (16) \end{aligned}$$

5 Hybrid Model

The hybrid model of the SATS–HVO concept extends the discrete state of the original model with the following continuous variables:

- A global current time t that evolves in a continuous way.
- For each aircraft A on final approach or missed approach

– the absolute time T_A when A initiated the approach, and
– the linear distance from its IAF, i.e., $D_A(t)$.

Discrete transitions are modified to handle the continuous variables as follows:

- *Approach initiation for vertical entry (left and right) and approach initiation for lateral entry (left and right).* An aircraft A may initiate the approach when it is the first aircraft in the landing sequence or its lead aircraft B is already on final approach, and

$$S_0 + L_b(iaf_B) - L_b(iaf_A) \quad \leq \quad D_B(t). \tag{17}$$

  In this case, the variable T_A is set to t.

- *Merging.* An aircraft A on the base segment turns to the final segment when it is the first aircraft in the landing sequence or its lead aircraft is already on final approach, and

$$D_A(t) \quad = \quad L_b(iaf_A). \tag{18}$$

- *Missed approach initiation.* An aircraft A on the final segment may go to the missed approach zone when it is the first aircraft in the landing sequence, and

$$D_A(t) \quad = \quad L_b(iaf_A) + L_f. \tag{19}$$

- *Landing.* An aircraft A on the final segment may land if it is the first aircraft in the landing sequence, there is no other aircraft on the runway, and

$$D_A(t) \quad = \quad L_b(iaf_A) + L_f. \tag{20}$$

- *Determination of lowest available altitude (left and right).* An aircraft A on missed approach may go to the holding fix at the lowest available altitude when

$$D_A(t) \quad = \quad L_b(iaf_A) + L_f + L_m(mahf_A). \tag{21}$$

  In this case, the variable T_A is reset to an undefined value.

Formulas (17)–(21) are derived from formulas (1)–(9).

In the next section we show that the hybrid model described here satisfies propositions 1 and 2.

6 Verification

The verification of the hybrid model uses a state exploration tool called Besc[2]. Besc was originally developed for the verification of the discrete model of the SATS–HVO concept [4]. Roughly speaking, Besc is a basic explicit model checker, written and formally verified in PVS.

In order to write the specification of the hybrid model as a discrete transition system, the continuous behavior has to be encoded symbolically. Therefore, the global state of the SCA is extended with a new field `eql`, which is a list of symbolic expressions known to be valid at a given discrete time. These symbolic expressions are formed by $\leq$-inequalities, additions, and continuous variables. The hybrid transition system described in Section 5 is encoded as follows:

- *Approach initiation for vertical entry (left and right) and approach initiation for lateral entry (left and right).* Let A be the aircraft that initiates the approach. The following symbolic properties are added to `eql`:

[2]Besc is available from http://research.nianet.org/~munoz/Besc.

– The fact that A is in the base segment, i.e,

$$T_A \leq t, \quad \text{and} \tag{22}$$
$$D_A(t) \leq L_b(iaf_A). \tag{23}$$

– If B is the lead aircraft of A, the fact that the aircraft are spaced at time T_A, i.e.,

$$T_B \leq T_A, \quad \text{and} \tag{24}$$
$$S_0 + L_b(iaf_B) \leq D_B(T_A) + L_b(iaf_A). \tag{25}$$

– For all aircraft C on missed approach,

$$L_b(iaf_A) + L_f \leq D_C(T_A). \tag{26}$$

- *Merging.* Let A be that aircraft that goes into the final segment. Formula (23) is removed from `eql` and the fact that A is on the final segment is added to `eql`:

$$D_A(t) \leq L_b(iaf_A) + L_f. \tag{27}$$

- *Missed approach initiation.* Let A be the aircraft that initiates the missed approach. Formula (27) is removed from `eql` and the fact that A is on missed approach is added to `eql`:

$$D_A(t) \leq L_b(iaf_A) + L_f + L_m(mahf_A). \tag{28}$$

- *Landing.* Let A be the aircraft that is landing. All the inequalities related to A are removed from `eql` except instances of formulas (24) and(25) when B, the previous lead aircraft of A, is on missed approach.

- *Determination of lowest available altitude (left and right).* Let A be the aircraft that goes to the lowest available altitude. All the inequalities related to A are removed from `eql`.

The resulting model is a finite discrete transition system that can be fully explored in PVS using Besc. Indeed, the explicit model checker reports 2768 reachable states and the diameter 27. To complete the verification, we consider all the states s in the set of reachable states:

- For each pair of aircraft A and B in s such that A and B are on the final approach and B is the lead of aircraft A, the following lemma is generated:

$$\texttt{eql}(s) \implies S_T \leq S_{A \to B}(t). \tag{29}$$

- For each pair of aircraft A and B in s such that they are on missed approach to the same fix and B is before A in the landing sequence, the following lemma is generated:

$$\texttt{eql}(s) \implies S_{MAZ} \leq S_{A \to B}(t). \tag{30}$$

Without counting repetitions, 117 lemmas were generated. From those, 73 lemmas are instances of Formula (29) and the remaining 44 lemmas are instances of Formula (30). The first set of lemmas corresponds to Proposition 1 and the second one corresponds to Proposition 2. All these lemmas were automatically proved in PVS using two custom made strategies.

7 Theoretical Spacing

This section describes the analytical derivation of formulas (11) and (12).

7.1 Spacing on final approach

Let A and B be aircraft on final approach at time t such that B is the lead of aircraft A. Since B is on final approach, by formulas (3) and (4),

$$D_B(t) \quad \leq \quad L_b(iaf_B) + L_f. \tag{31}$$

Using Formula (8) on A and B yields

$$S_0 + L_b(iaf_B) - L_b(iaf_A) \quad \leq \quad D_B(T_A). \tag{32}$$

Subtracting formulas (31) and (32), we get

$$D_B(t) - D_B(T_A) \quad \leq \quad L_b(iaf_A) + L_f - S_0. \tag{33}$$

Formula (2) states that $T_A \leq t$. Therefore, using Formula (6) on A and B,

$$(t - T_A)v_{\min} \quad \leq \quad D_B(t) - D_B(T_A), \quad \text{and} \tag{34}$$

$$D_A(t) - D_A(T_A) \quad \leq \quad (t - T_A)v_{\max}. \tag{35}$$

But, $D_A(T_A) = 0$, by Formula (1). Therefore,

$$D_A(t) \quad \leq \quad (t - T_A)v_{\max}. \tag{36}$$

From formulas (33) and (34),

$$t - T_A \quad \leq \quad \frac{L_b(iaf_A) + L_f - S_0}{v_{\min}}. \tag{37}$$

Hence, the theoretical spacing for two aircraft on final approach is given by

$$\begin{aligned}
S_{A\to B}(t) &= D_B(t) - D_A(t) + L_b(iaf_A) - L_b(iaf_B) \\
&= D_B(T_A) + (D_B(t) - D_B(T_A)) - D_A(t) + L_b(iaf_A) - L_b(iaf_B) \\
&\geq S_0 + (D_B(t) - D_B(T_A)) - D_A(t), \quad \text{by Formula (32)}, \\
&\geq S_0 + (t - T_A)v_{\min} - (t - T_A)v_{\max}, \quad \text{by formulas (34) and (36)}, \\
&\geq S_0 - (L_b(iaf_A) + L_f - S_0)\frac{v_{\max} - v_{\min}}{v_{\min}}, \quad \text{by Formula (37)}, \\
&\geq S_0 - (L_{max} + L_f - S_0)\Delta_v, \quad \text{by formulas (14) and (16)}.
\end{aligned}$$

The latter expression defines S_T as given in Formula (11).

7.2 Spacing on missed approach

Let A and B be aircraft on missed approach at time t such that B is before A in the landing sequence. Since B is on missed approach, by Formula (5),

$$D_B(t) \quad \leq \quad L_b(iaf_B) + L_f + L_m(mahf_B). \tag{38}$$

We consider two cases:

- When A initiated the approach, B was on missed approach. Using Formula (9) on A and B yields

  $$L_b(iaf_B) + L_f \quad \leq \quad D_B(T_A). \tag{39}$$

  Subtracting formulas (38) and (39), we get

  $$D_B(t) - D_B(T_A) \quad \leq \quad L_m(mahf_B). \tag{40}$$

Formulas (34)–(36) are derived as in Section 7.1. From formulas (34) and (40),

$$t - T_A \quad \leq \quad \frac{L_m(mahf_B)}{v_{\min}}. \tag{41}$$

Hence, the theoretical spacing in this case is given by

$$\begin{aligned}
S_{A\to B}(t) &= D_B(t) - D_A(t) + L_b(iaf_A) - L_b(iaf_B) \\
&= D_B(T_A) + (D_B(t) - D_B(T_A)) - D_A(t) + L_b(iaf_A) - L_b(iaf_B) \\
&\geq L_b(iaf_A) + L_f + (D_B(t) - D_B(T_A)) - D_A(t), \quad \text{by Formula (39)}, \\
&\geq L_b(iaf_A) + L_f + (t - T_A)v_{\min} - (t - T_A)v_{\max}, \quad \text{by formulas (34) and (36)}, \\
&\geq L_b(iaf_A) + L_f - L_m(mahf_B)\frac{v_{\max} - v_{\min}}{v_{\min}}, \quad \text{by Formula (41)}, \\
&\geq L_{min} + L_f - L_{maz}\Delta_v, \quad \text{by formulas (13), (15), and (16)}.
\end{aligned}$$

- When A initiated the approach, aircraft B and X where on final approach, B was the lead of aircraft X, and X was the lead aircraft of A. Using Formula (9) on A, X, and B, yields

$$S_0 + L_b(iaf_B) - L_b(iaf_X) \quad \leq \quad D_B(T_X), \quad \text{and} \tag{42}$$
$$S_0 + L_b(iaf_X) - L_b(iaf_A) \quad \leq \quad D_X(T_A). \tag{43}$$

Subtracting formulas (38) and (42), we get

$$D_B(t) - D_B(T_X) \quad \leq \quad L_b(iaf_X) + L_f + L_m(mahf_B) - S_0. \tag{44}$$

Formula (36) is derived as in Section 7.1. From Formula (7), $T_X \leq T_A$, and from Formula (1), $D_X(T_X) = 0$. Therefore, using Formula (6) on X,

$$D_X(T_A) \quad \leq \quad (T_A - T_X)v_{\max}. \tag{45}$$

From Formula (2), $T_X \leq t$. Using Formula (6) on B,

$$(t - T_X)v_{\min} \quad \leq \quad D_B(t) - D_B(T_X). \tag{46}$$

From formulas (44) and (46),

$$t - T_X \quad \leq \quad \frac{L_b(iaf_X) + L_f + L_m(mahf_B) - S_0}{v_{\min}}. \tag{47}$$

Hence, the theoretical spacing in this case is given by

$$\begin{aligned}
S_{A\to B}(t) &= D_B(t) - D_A(t) + L_b(iaf_A) - L_b(iaf_B) \\
&= D_B(T_X) + (D_B(t) - D_B(T_X)) - D_A(t) + L_b(iaf_A) - L_b(iaf_B) \\
&\geq S_0 + L_b(iaf_A) - L_b(iaf_X) + (D_B(t) - D_B(T_X)) - D_A(t), \\
&\quad \text{by Formula (42)}, \\
&\geq S_0 + L_b(iaf_A) - L_b(iaf_X) + (t - T_X)v_{\min} - (t - T_A)v_{\max}, \\
&\quad \text{by formulas (36) and (46)}, \\
&= S_0 + L_b(iaf_A) - L_b(iaf_X) - (t - T_x)(v_{\max} - v_{\min}) + (T_A - T_X)v_{\max} \\
&\geq S_0 + L_b(iaf_A) - L_b(iaf_X) - (t - T_x)(v_{\max} - v_{\min}) + D_X(T_A), \\
&\quad \text{by Formula (45)}, \\
&\geq 2S_0 - (t - T_x)(v_{\max} - v_{\min}), \quad \text{by Formula (43)}, \\
&\geq 2S_0 - (L_b(iaf_X) + L_f + L_m(mahf_B) - S_0)\frac{v_{\max} - v_{\min}}{v_{\min}}, \\
&\quad \text{by Formula (47)}, \\
&\geq 2S_0 - (L_{max} + L_f + L_{maz} - S_0)\Delta_v, \quad \text{by formulas (14), (15), and (16)}.
\end{aligned}$$

In both cases,

$$S_{A\to B}(t) \quad \geq \quad \min(L_{min} + L_f - L_{maz}\Delta_v, 2S_0 - (L_{max} + L_f + L_{maz} - S_0)\Delta_v).$$

This expression defines S_{MAZ} as given in Formula (12). Note that

$$S_{MAZ} \quad = \quad 2S_0 - (L_{max} + L_f + L_{maz} - S_0)\Delta_v, \tag{48}$$

when

$$1 + \frac{v_{\min}}{v_{\max}} \quad \leq \quad \frac{L_{min} + L_f}{S_0}. \tag{49}$$

Furthermore,

$$S_t \quad \leq \quad S_{MAZ}, \tag{50}$$

when

$$L_{maz}\Delta_v \quad \leq \quad S_0. \tag{51}$$

8 Conclusion

We have presented analytical formulas to compute the minimum spacing on final approach and missed approach for the SATS–HVO operational concept. Using theorem proving and model checking techniques, we have exhaustively explored the set of nominal operations and mechanically verified the spacing properties.

From a practical point of view, the results presented in this paper can be used to configure a nominal SCA and the parameters of the baseline procedure for self separation. For instance, consider a symmetrical nominal SCA where $L_b(left) = L_b(right) = 5$ nm, $L_f = 10$ nm, and $L_m(left) = L_m(right) = 13$ nm. If the initial separation S_0 is 6 nm and $v_{\min} = 90$ kt, $v_{\max} = 120$ kt, then

$$\begin{aligned} L_{min} = L_{max} &= 5 \text{ nm}, \\ L_{maz} &= 13 \text{ nm}, \quad \text{and} \\ \Delta_v = \frac{120 - 90}{90} &= \frac{1}{3}. \end{aligned}$$

The value of S_T is computed using Formula (11):

$$S_T = 6 - \frac{5 + 10 - 6}{3} \quad = \quad 3 \text{ nm}.$$

This configuration of the SCA satisfies Formula (49). Therefore, the value of S_{MAZ} can computed using Formula (48):

$$S_{MAZ} = 12 - \frac{5 + 10 + 13 - 6}{3} \quad = \quad 4.66 \text{ nm}.$$

Hence, if the initial spacing of the trail aircraft with respect to the lead aircraft is 6 nm, the SATS–HVO concept of operations guarantees a minimum spacing of 3 nm on final approach and 4.66 nm on missed approach.

The geometrical analysis used in this paper can be extended to study Euclidean separation of aircraft on final approach and missed approach. Figure 4 illustrates a nominal SCA where aircraft on missed approach turn toward their missed approach zone α degrees with respect to the runway, fly a straight trajectory of M nautical miles, and then turn to their MAHF. A geometrical analysis reveals that

$$M \quad = \quad \frac{\min(S_T, S_{MAZ})}{2} \tag{52}$$

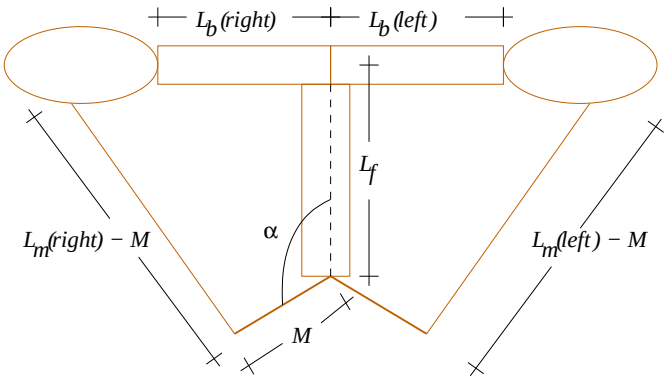


Figure 4: Nominal SCA

achieves maximum separation for an arbitrary α. In this case, the minimum Euclidean distance D_α that the concept guarantees for an aircraft on final approach and an aircraft on missed approach is given by

$$D_\alpha \quad = \quad M\sqrt{2(1-\cos\alpha)}. \tag{53}$$

In the example above, the optimal value of M, given by Formula (52), is 1.5 nm. The minimum Euclidean distance between an aircraft on final approach and an aircraft on missed approach, for different values of α, is computed using Formula (53):

- $D_{60^o} = 1.5$ nm.
- $D_{90^o} = 2.12$ nm.
- $D_{120^o} = 2.59$ nm.

Increasing the initial spacing S_0 to 7 nm yields the following values: $S_T = 4.33$ nm, $S_{MAZ} = 7$ nm, $M = 2.16$ nm, $D_{60^o} = 2.16$ nm, $D_{90^o} = 3.06$ nm, and $D_{120^o} = 3.75$ nm.

The mechanical verification is necessary to make sure that no cases were forgotten on the theoretical derivation presented in Section 7. For instance, the case analysis on Section 7.2 was discovered by direct inspection on the lemmas automatically generated from the set of reachable states. Formal proofs are the ultimate guarantee that the mathematical development presented here is correct.

References

[1] T. Abbott, K. Jones, M. Consiglio, D. Williams, and C. Adams. Small Aircraft Transportation System, High Volume Operation concept: Normal operations. Technical Report NASA/TM-2004-213022, NASA Langley Research Center, NASA LaRC Hampton VA 23681-2199, USA, 2004.

[2] G. Dowek, C. Muñoz, and V. Carreño. Abstract model of the SATS concept of operations: Initial results and recommendations. Technical Report NASA/TM-2004-213006, NASA Langley Research Center, NASA LaRC,Hampton VA 23681-2199, USA, March 2004.

[3] *Federal Aviation Regulations/Aeronautical Information Manual*, 1999.

[4] C. Muñoz, G. Dowek, and V. Carreño. Modeling and verification of an air traffic concept of operations. *Software Engineering Notes*, 29(4):175–182, 2004. A long version appears as report NASA/TM-2004-213006.

[5] SATS Program Office. Small aircraft transportation system program plan. http://sats.larc.nasa.gov/documents.html, 2001.

[6] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

Certification of Complex Systems

Nicholas Tudor
QinetiQ

The cost of testing in certification is significant; furthermore, there is evidence that re-testing as designs mature can be up to 50% of the original test cost with obvious consequences for the whole software life cycle cost. Testing is particularly expensive in re-certification where test and test evidence has to be re-factored or even discarded. Test evidence can be somewhat subjective in that not all circumstances are examined. Through the use of formal methods and, in particular, formal proof it should be possible to reduce the extent of verification testing but still achieve at least the same assurance as test evidence in a more objective manner. This presentation firstly assesses the extent of testing that is necessary to achieve satisfactory evidence for certification and then shows that formal methods can subsume some of that need in certifying systems. This comparison is presented in the form of a structured argument appropriate for the certification of the software in a system to RTCA DO178B Level A. Typically, formal methods have been thought of as slow, costly and inapplicable to industrial scale projects. It is now possible under certain circumstances to formally derive code from a Model Based Design automatically and, through the use of mechanical proof techniques, show that the code implements the design. If these techniques were then to be applied to typical Level A development, there can be expected to be cost savings. The pen-ultimate part of this presentation will outline the typical costs a typical development project and then, using the ClawZ toolset as an exemplar, compare the same development using the automated formal techniques. Finally, there remains the question of validation and confirmation that requirements are complete and consistent. Whilst tools like CLawZ can assist, there always remains some continuous domain to be checked. These are the challenges posed at the end of the presentation.